UNIVERSITÁ DI PISA
Facoltá di Ingegneria

Corso di laurea specialistica in Ingegneria Informatica
Curriculum Networking e Multimedia

PROGETTAZIONE E REALIZZAZIONE DI UN SISTEMA PER L'ACQUISIZIONE PARTECIPATIVA DI INFORMAZIONI RELATIVE ALLA QUALITA' DEL SEGNALE IN UNA RETE DI TELEFONIA MOBILE

Tesi di

Gabriele Giovenco

Relatori

Prof. Marco Avvenuti .....

Prof. Alessio Vecchio .....

Candidato

Gabriele Giovenco .....

Sessione Laurea 10 Maggio 2012
Anno Accademico 2011/2012
Consultazione consentita

II

A mia madre e mio padre,
a mia sorella,
ai miei nonni
e al mio amore
che mi hanno dato
la possibilitá
e la forza di provarci.

Sommario

L'evoluzione tecnologica di questo periodo storico ha permesso l'ingresso di dispositivi elettronici sempre piú all'avanguardia all'interno del mercato dell'informatica. Grazie alla loro crescente "intelligenza", questi nuovi terminali sono in grado di influenzare la vita quotidiana di tutte le persone, al punto tale da considerare l'utente non piú come una semplice entitá che utilizza i dati, ma piuttosto come un'entitá capace di fornire i dati stessi, da riutilizzare per fornire nuovi servizi di comune interesse. In commercio è possibile acquistare uno di questi dispositivi anche a prezzi ragionevoli; negli ultimi anni questo fatto ha scatenato una grandissima diffusione di un particolare tipo di dispositivo mobile: lo *smartphone*. Questo, grazie alla sua ormai infinita capacitá di calcolo e ad una connessione dati permanente, potrebbe essere in grado di sostituire in tutto e per tutto un personal computer. Si stima infatti che nel giro di 3/4 anni, gli smartphones (oggi circa 500 milioni nel mondo) riescano a superare il numero dei pc (attualmente pari a circa 1.4 miliardi). Grazie a questa fitta copertuta territoriale e grazie al fatto che all'interno degli smartphone sono assemblati svariati tipi di sensori, è nata una nuova classe di applicazioni sensibili all'ambiente. Tali applicazioni stanno alla base dei cosiddetti progetti di *urban sensing*. All'interno di questa classe si trovano progetti che, sfruttando dati ambientali, hanno il compito di analizzare problematiche di carattere urbano. Il presente lavoro di tesi consiste quindi nella realizzazione di un'applicazione che, dopo aver raccolto un particolare tipo di dati ambientali provenienti dagli smartphones, permetta ad un utente di poter effettuare alcune analisi riguardo l'andamento del livello di ricezione dei telefoni cellulari all'interno di un'area geografica. Il dato fondamentale che permette di ottenere un indicazione sull'andamento del segnale è il parametro RSSI (Received Signal Strength Indication), il quale viene raccolto dai singoli smartphones con piattaforma *Android* correttamente registrati a questo particolare servizio. La componente dell'architettura che si occupa della gestione dei sensori è *cloudsensor*: tale sistema permette di trattare come una comune rete di sensori l'insieme degli smartphones che si sono resi disponibili nei confronti del servizio. L'applicazione da installare sui terminali android per permettere l'interazione con cloudsensor si chiama *CSAMS* (*CloudSensor Acquisition & Management System*). Prima di procedere alla realizzazione dell'applicazione web si è rivelato necessario apportare le dovute modifiche all'applicazione mobile, in quanto tale applicazione non supportava il sensore utilizzato durante il lavoro di questa tesi.

Abstract

The technological evolution of this historical period has allowed the entry of more and more advanced electronic devices into the IT market. Thanks to their growing "intelligence", these new terminals are able to influence the daily lives of all people and the user is no longer considered as a simple entity that uses the data, but rather as an entity that provide data, in order to reuse the data to provide new services of common interest. In the market you can buy one of these devices with a reasonably price and in these years this fact has sparked an huge spread of a particular type of mobile device: the *smartphone*. This kind of device, with its now infinite computing capacity and with a permanent data connection, may be able to replace in all aspects a personal computer. Is estimated that within 3/4 years, smartphones (currently about 500 million worldwide) are able to exceed the number of PCs (currently approximately 1.4 billion). Thanks to this dense coverage and thanks to the fact that in all smartphones are assembled many kinds of sensors, a new class of environmentally sensitive applications has been created. These kind applications is the basis of so-called *urban sensing projects*. Within this class of applications there are projects that, by using environmental data, have the task of analyzing problems of urban interest. The present work consists to design an application that, after collecting a particular type of environmental data from smartphones, make a user to be able to do some analysis about the distribution of the level of mobile phone reception in a geographical area. The main parameter that allows to obtain an indication of the progress of the signal parameter is the RSSI (Received Signal Strength Indication), which is collected by every smartphones with *Android* platform. The component of the architecture that is responsible to manage sensors is *cloudsensor*: this system allows to handle a large number of smartphones like a common network of sensors. The application that must be installed on the Android terminals to allow interactions with cloudsensor, is called *CSAMS* (*CloudSensor Acquisition & Management System*). Before the creation of the web application has been necessary to make appropriate modifications on the mobile application, since this application did not support the sensor used during the work of this thesis.

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'acronimo RSSI significa "Received Signal Strenght Indicator" e rappresenta un parametro fondamentale nel campo delle telecomunicazioni; con tale indicatore si fa riferimento al livello di potenza del segnale presente sul lato ricevitore e la qualitá di una conversazione telefonica è direttamente proporzionale al valore di tale parametro. Ci sono diversi fattori che possono influire sulla potenza del segnale ricevuto, come ad esempio la lontananza dall'antenna alla quale si è collegati, la presenza di ostacoli come palazzi o gallerie etc... Generalmente, la zona dove si ha una maggiore probabilitá di trovare un "buon" segnale è quella urbana, anche se in questi posti alle volte si possono trovare zone nelle quali si ha una cattiva ricezione: le cosiddette "dead zones".

Diventa quindi interessante cercare di monitorare il comportamento del parametro RSSI in tutto il territorio, in modo da individuare aree con bassa ricezione ed andare ad ovviare ad eventuali problemi laddove possibile; oppure, semplicemente, potrebbe essere interessate osservare come si comporta questo parametro in funzione del tempo. Ecco le motivazioni che stanno alla base di questo lavoro di tesi: si cerca quindi di sfruttare la proprietá intrinseca di mobilitá di dispositivi, come i moderni smartphone, per utilizzarli non come semplici telefoni cellulari ma piuttosto come sensori capaci di misurare in mobilitá un particolare parametro di interesse. La domanda che si potrebbe porre è la seguente: il parametro monitorato da questa tipologia di sensori, viene misurato con una precisione accettabile? Se si pensa che in questo periodo storico una grandissima parte della popolazione possiede ormai uno smartphone con diversi sensori integrati, viene spontaneo rispondere alla domanda precedente in modo positivo. L'accuratezza del parametro misurato dipende fortemente dalle misurazioni effettuate sullo stesso: avere

Figura 1.1: Sensori integrati

a disposizione numerosi smartphone significa avere potenzialmente numerose misurazioni, che a sua volta significa avere una buona precisione sul parametro misurato. I dati prelevati da ogni tipo di sensore vengono memorizzati all'interno di un database ed è possibile quindi andare a prelevarli per monitorarne il comportamento, in modo da poter fornire un servizio che possa interessare ad un vasto numero di utenti. Tutti gli smartphone dispongono di svariati tipi di sensori al loro interno; nel caso particolare di questa tesi si sono sfruttati il sensore che rileva il parametro RSSI, un sensore che rileva l'id della cella alla quale il telefono è collegato ed infine il sensore GPS che è in grado di fornire le coordinate geografiche attuali con un certo livello di accuratezza (vedi Figura 1.1).

Se si vuole pensare ad un possibile scenario, ci si puó immaginare uno smartphone in movimento verso una certa direzione; grazie ad una applicazione installata sul terminale, il dispositivo si pone in attesa di eventuali richieste di tasking (con tale termine si fa riferimento ad una procedura di sensing da parte di uno smartphone che ha ricevuto una richiesta). Quando una richiesta arriva, lo smartphone comincia a prelevare le informazioni sui parametri di interesse, inserendo inoltre un timestamp e spedendo il tutto ad un Data Collection Server, il quale si preoccupa di memorizzare in un database le osservazioni ricevute (in Figura 1.2 viene mostrata l'architettura di base).

Lo scopo di lavoro di questa tesi è la realizzazione del lato applicativo dell'architettura mostrata in figura, ovvero la creazione di una web application, chiamata RSSIWatcher, che si interfaccia con il Data Col-

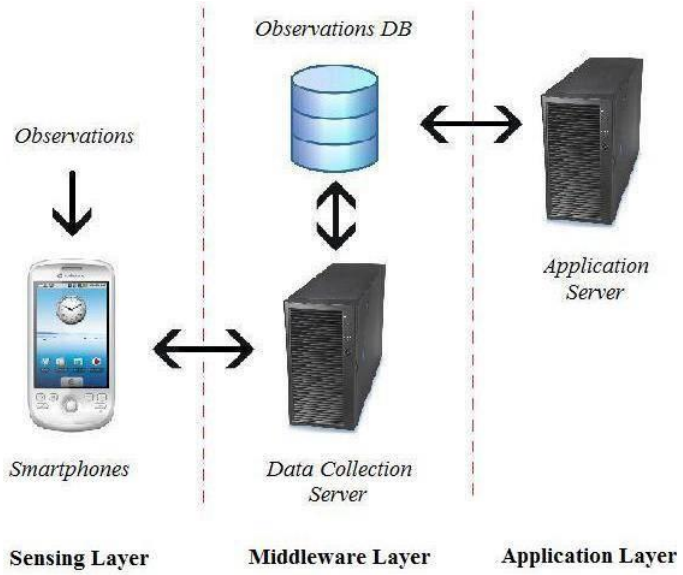

Figura 1.2: Architettura base

lection Server sopra citato. Si è inoltre rivelato necessario effettuare le dovute modifiche all'applicazione CSAMS da installare sul terminale mobile, in quanto, il sensore utilizzato per monitorare il parametro RSSI, non era supportato nella sua versione iniziale. RSSIWatcher è l'applicazione che permette di mostrare su una mappa le osservazioni effettuate dagli smartphone durante i loro task; fornisce inoltre, all'utente che la sta utilizzando, la possibilitá di consultare la lista dei terminali attualmente registrati al servizio e quindi la possibilitá di iniziare un nuovo task specificando l'id del terminale-sensore, la durata dell'osservazione ed il periodo di campionamento. Inoltre, la web application permette di specificare un'area circolare della mappa e di mostrare, in un grafico di tipo scatter, come si è comportato il parametro RSSI all'interno di tale area in funzione del tempo.

1.1 Stato dell'arte

In questa sezione vengono introdotti alcuni argomenti di ricerca che si sono approfonditi durante il progetto della tesi. Grazie a questi concetti è possibile capire piú facilmente in quale settore si colloca il lavoro svolto.

1.1.1 Smartphones e reti di sensori

Attualmente si sente spesso parlare di Wireless Sensor Network: questo particolare tipo di rete è costituito da un insieme di dispositivi elettronici autonomi (sensori) che sono in grado di prelevare dati dall'ambiente in cui si trovano e di comunicarli altrove.

Stando alle caratteristiche dei moderni terminali si puó trarre la conclusione che una rete di smartphone, organizzata in modo appropriato, puó essere in grado di sostituire in tutto e per tutto una WSN garantendo una buona accuratezza del parametro monitorato. Come giá detto, la diffusione di smartphone di ultima generazione è in forte crescita.

1.1.2 Mobile Phone Sensing

La presenza di differenti tipi di sensori integrati in uno smartphone, fa si che si possa introdurre questo dispositivo all'interno di nuovi contesti come ad esempio il monitoraggio di parametri fisiologici, oppure il monitoraggio di parametri ambientali (come ad esempio il livello di carbonio presente nell'aria o come in questo caso il valore del parametro RSSI), dando vita a un nuovo ramo di ricerca chiamato Mobile Phone Sensing. Questa nuova area di ricerca si sta rivelando molto vasta e vi si trovano molteplici campi applicativi grazie ad una importante proprietá degli smartphone: la programmabilitá. Aggiungendo alla capacitá di sensing anche le risorse computazionali e di comunicazione, si ottiene il dispositivo che sta alla base del Mobile Phone Sensing; un dispositivo dove è possibile installare applicazioni create ad hoc. Un altro aspetto fondamentale di questo campo di ricerca è la possibilitá di creare dei sistemi di back-end capaci di affiancare i dispositivi mobili con ulteriori risorse di calcolo e di storage dei dati rilevati. In questo lavoro di tesi si è fatto utilizzo di un sistema del genere per ottenere una web application capace di fornire informazioni utili basandosi su una ingente quantitá di dati precedentemente memorizzati.

Le applicazioni che si trovano all'interno del Mobile Phone Sensing sono applicazioni di tipo context-aware: ovvero sono in grado di fornire risultati che dipendono dalle informazioni di contesto rilevate dagli smartphone, come in questo caso dati ambientali (questo tipo di applicazioni verrá descritto meglio nella prossima sottosezione). In base al numero di persone che sono interessate ai servizi offerti da queste applicazioni, si possono distinguere tre diversi livelli di scala:

- **personal sensing**: sono applicazioni designate all'uso del singolo possessore dello smartphone. I dati prodotti dal dispositivo quindi

non devono essere condivisi con altri individui. Un semplice esempio sono le classiche applicazioni che tengono traccia del proprio allenamento quotidiano.

- **group sensing**: fanno parte di questa categoria quelle applicazioni dove piú individui partecipano al sensing, in quanto condividono un obiettivo comune. In questo caso è presente un elemento di fiducia reciproca: si da per scontato infatti che i dati recuperati siano credibili, in quanto si ha un certo livello di controllo sulle persone che partecipano al sensing.

- **community sensing**: sono applicazioni dove viene trattata una grande quantitá di dati. L'analisi e la condivisione delle informazioni sono a disposizione del bene della comunitá. A differenza della precedente categoria, ci si trova di fronte a una situazione dove viene meno la fiducia reciproca in quanto i possessori degli smartphone possono non conoscersi e quindi possono non avere gli stessi interessi nei confronti dell'applicazione. La precisione del dato misurato in questo caso è direttamente proporzionale al numero dei partecipanti che contribuiscono al sensing.

Si puó fare una ulteriore classificazione delle applicazioni di Mobile Phone Sensing in base al livello di controllo che ha il possessore dello smartphone sulla partecipazione al sensing. In questo caso le categorie sono due:

- **partecipatory sensing**: l'utente ha il pieno controllo sulla collezione dei dati e puó decidere come, su cosa e dove effettuare il sensing.

- **opportunistic sensing**: in questo caso la collezione dei dati è completamente automatizzata e non comporta il coinvolgimento dell'utente che, una volta avviata l'applicazione che permette di fare il recupero dei dati, non ha il controllo sulle informazioni prodotte.

Questi due paradigmi presentano pro e contro. Il partecipatory sensing presenta il vantaggio che l'utente, avendo un ruolo molto attivo sul sensing, puó evitare di inviare informazioni non utilizzabili o inutili per il sistema. Di contro questo paradigma presenta lo svantaggio che l'utente, proprio perche ha un ruolo attivo sul sensing, deve essere a conoscenza delle finalitá del servizio, in modo che sia in grado di settare in maniera opportuna il proprio dispositivo. L'opportunistic sensing assolve l'utente da qualsiasi compito decisionale e quindi si ha un distacco notevole dal

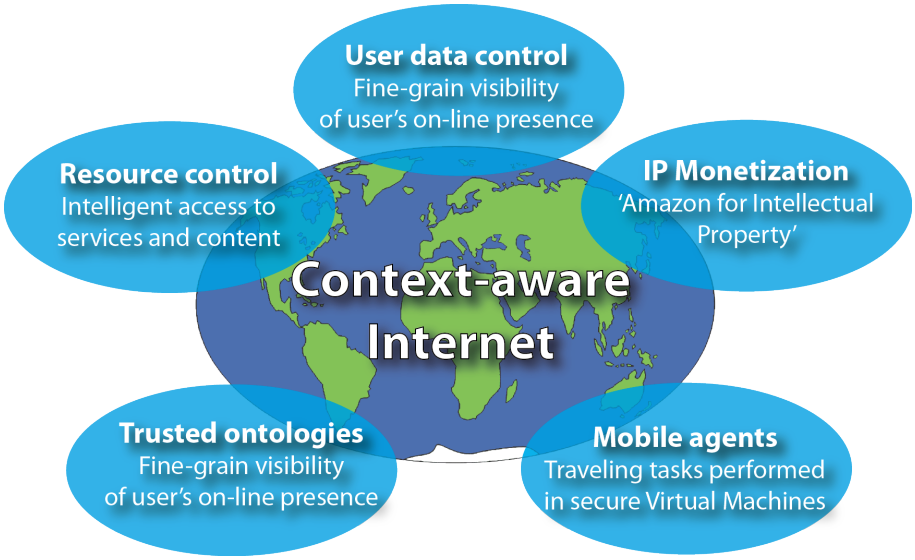

Figura 1.3: Servizio context-aware

servizio; in questo modo si garantisce un flusso continuo di dati verso il database che li ospita. Questa soluzione risulta quindi molto vantaggiosa qualora si abbia a che fare con servizi che presentano particolari vincoli temporali. Di contro peró, questo paradigma rischia di produrre un overhead non trascurabile di dati che non risultano utili alle finalitá del servizio, ad esempio quando uno smartphone per qualche motivo non debba comportarsi da provider di informazioni.

Di nuovo si possono distinguere altre categorie di applicazioni di Mobile Phone Sensing in base, questa volta, a come si comportano i dispositivi nei confronti della memorizzazione e del trattamento dei dati appena rilevati. Esistono infatti applicazioni mobili che memorizzano e raffinano i dati prelevati dal contesto prima di inviarli ad un server; insieme a queste esistono anche applicazioni mobili che, appena prelevano il dato grezzo dal contesto, lo inviano incondizionatamente al server.

In questa tesi, l'applicazione mobile presente sul dispositivo spedisce istantaneamente il dato prelevato al server ed inoltre offre la possibilitá di settare alcuni parametri dell'osservazione. Puó essere definita quindi come una applicazione di tipo partecipatory sensing e community sensing, in quanto l'interesse del servizio puó essere allargato ad un numero molto elevato di individui.

1.1.3 Servizi context-aware

Si definiscono applicazioni context-aware quelle applicazioni che sono in grado di comportarsi in maniera diversa in base alle informazioni che ottengno dal contesto nel quale sono immerse; come ad esempio la posizione dell'utente (vedi Figura 1.3). In questa categoria di servizi ci sono due entitá principali:

- **context consumer**: entitá che modificano il comportamento globale dell'applicazione dopo aver ricevuto le informazioni di contesto necessarie.

- **context provider**: entitá che producono informazioni di contesto utilizzate dai context consumer.

Anche in questo caso, è possibile elencare diverse categorie di applicazioni context-aware in base alla modalitá di comunicazione tra context provider e context consumer:

- **Direct Access to Sensor**: situazione in cui piú providers e piú consumers comunicano in base alle proprie necessitá. In questa categoria i sensori si trovano all'estremo della rete e collezionano dati ambientali.

- **Middleware Infrastructure**: si introduce un middleware che si comporta come unico context-provider. In questo modo si ha un netto disaccoppiamento tra le entitá che forniscono ad esso informazioni di contesto (i sensori) e i consumer che, per mezzo di API e di operazioni a loro consentite dal sistema, utilizzano il middleware stesso per recuperare le informazioni di contesto.

- **context server**: in questo caso si ha un unico modulo software che si comporta da consumer e che comunica con piú provider dai quali riceve informazioni di contesto. Queste informazioni vengono restituite alle applicazioni clienti che le richiedono e quindi vedono come unica sorgente dei dati questo modulo software.

Se si pensa alle finalitá che possono avere le applicazioni context-aware è allora possibile effettuare un ulteriore classificazione:

- **context information source**: questo tipo di applicazioni si limita a presentare ai clienti che le richiedono le informazioni di contesto memorizzate ed inviate dal context provider.

- **context synthesis services**: queste applicazioni effettuano elaborazioni sui dati prima di restituirli ai clienti, in questo modo forniscono informazioni piú complesse dei semplici dati grezzi.

Nel caso di questo lavoro di tesi, dove la web application costruita è inserita in un sistema costituito da varie componenti, si individuano come context-providers i terminali mobili che prelevano informazioni sul parametro RSSI accompagnate da altre informazioni riguardanti posizione, id di cella e da una marca temporale; il context consumer invece è il modulo software installato sul server e si occupa dello storage dei dati ricevuti dai providers. La comunicazione tra consumer e providers è garantita grazie a un apposito strato middleware. Il context consumer è una applicazione di tipo context information source, ovvero si limita a rendere noti al cliente i dati raccolti dai providers senza effettuarne alcuna elaborazione; tale applicazione cliente è proprio la web application progettata in questo lavoro di tesi, ovvero RSSIWatcher.

Capitolo 2

Tecnologie di riferimento

In questo capitolo viene illustrato il funzionamento di tutti i componenti software utilizzati durante il lavoro di tesi. Verranno analizzati quindi:

- **lo stack software Android di Google** in quanto la programmazione android è stata utilizzata per ampliare le funzionalitá dell'applicazione mobile da installare sugli smartphone per supportare il nuovo sensore.
- **il framework Sensor Web Enablement** messo a disposizione dal consorzio 52North, in quanto si sono utilizzate tali librerie messe a disposizione da questo framework per implementare l'applicazione web.
- **il sistema CloudSensor** che si occupa dell'acquisizione dei dati provenienti dai dispositivi mobili.

2.1 Android

Android è uno stack software composto da: un sistema operativo, un middleware, alcune applicazioni native e alcune librerie API utili allo sviluppo di nuove applicazioni. Nel recente passato si trovavano telefoni cellulari aventi sistemi operativi fortemente chiusi; tali sistemi operativi richiedevano ambienti di sviluppo proprietari, privilegiando di conseguenza le applicazioni native a scapito di quelle scritte da terze parti. Questo fatto ha rappresentato un vero e proprio ostacolo per lo sviluppatore, il quale non poteva sfruttare la crescente potenza hardware di questi terminali per sviluppare applicazioni piú avanzate.

Android è un ambiente ottimale per lo sviluppo di applicazioni mobili, in quanto offre la possibilitá di sfruttare tutto l'hardware presente nel dispositivo grazie all'utilizzo di funzioni dedicate: è infatti possibile accedere ed utilizzare tutti i sensori presenti nel terminale, registrare audio e video, rilevare le coordinate geografiche attuali, effettuare comunicazioni tra applicazioni diverse, creare database all'interno del terminale stesso, sfruttare la connessione dati per comunicare con server remoti etc...

2.1.1 La storia di Android

Nel passato, i linguaggi di programmazione piú usati per creare applicazioni sono stati linguaggi del tipo C o C++. Chi si trovava a che fare con tali linguaggi di programmazione, aveva spesso la necessitá di conoscere bene l'hardware sul quale avrebbero dovuto girare le proprie applicazioni.

In seguito sono state create piattaforme come Symbian che permettevano allo sviluppatore di sfruttare meglio l'hardware del terminale, mettendo a disposizione delle API ancora di difficile utilizzo a causa della complessitá del linguaggio di programmazione utilizzato (C o C++).

Ecco quindi che si assiste all'introduzione delle MIDlet Java. Una Midlet JAVA è un'applicazione creata per sistemi embedded ed in particolare per i sistemi sui quali è installata una J2ME Virtual Machine: un processo capace di astrarre tutto l'hardware sottostante. Il vantaggio piú importante, ottenuto dall'utilizzo di questo tipo di tecnologia, è che tutte le applicazioni create possono girare su una vasta gamma di dispositivi che abbiano come unico requisito comune il supporto a Java. Anche in questo caso, peró, si trova ancora uno svantaggio determinante: la possibilitá di accedere all'hardware del dispositivo appare ancora molto ristretta. Infatti, questo tipo di applicazioni si limitavano ad essere utilizzate come normali programmi che non avevano la necessitá di sfruttare le potenzialitá hardware presenti nel dispositivo. Questa non era ancora considerata un idea sbagliata: era ritenuto normale, infatti, che solamente le applicazioni native e proprietarie potessero avere il diritto di sfruttare a pieno l'hardware del terminale e non si pensava ancora all'eventualitá che uno sviluppatore potesse creare applicazioni capaci di farlo.

Ecco quindi che lo step successivo è stato la nascita di Android: un sistema operativo capace di offrire allo sviluppatore la possibilitá di sfruttare a pieno la crescente potenza dell'hardware dei moderni smart-

phone. Come Android, esistono anche altri sistemi di questo tipo, come ad esempio Apple iPhone o Windows Mobile; ma sono tutte soluzioni ancora proprietarie che quindi, in qualche modo, tendono a privilegiare le appliacazioni native, a limitare la diffuzione di applicazioni di terze parti e quindi a limitare anche la comunicazione tra quest'ultime e i dati del dispositivo. Android invece, avendo un kernel Linux open-source, consente a tutte le applicazioni native e non, di muoversi e di sfruttare a 360 gradi tutte le potenzialitá del dispositivo, senza particolari limitazioni. Le applicazioni che girano su terminali Android hanno quindi tutte gli stessi diritti e fanno uso delle stesse API messe a disposizione dall'ambiente; tali applicazioni girano dunque in concorrenza ed è supportata la comunicazione fra esse. Android offre inoltre la possibilitá di far girare applicazioni anche in backgroud; quindi l'applicazione puó lavorare anche se l'utente sta utilizzando il proprio dispositivo per fare altre cose, come ad esempio una conversazione telefonica. L'applicazione quindi puó porsi in attesa di un evento in grado di svegliarla. Diverse applicazioni inoltre, possono essere in grado di scambiarsi messaggi di varia natura e possono anche condividere dati comuni; per prevenire ovvi rischi legati a questa caratteristica, tutti i dati utilizzati da una applicazione rimangono privati fino a che non si specifica esplicitamente il desiderio di condividerli, grazie a un meccanismo basato su permessi.

Tutte questi vantaggi hanno fatto ricadere la scelta sull'ambiente di sviluppo Android per lo sviluppo dell'applicazione lato mobile del sistema.

2.1.2 Lo stack Android

Verrá adesso esaminato nel dettaglio lo stack Android, descrivendo brevemente le parti che lo costituiscono. La struttura (vedi 2.1) è stratificata per livelli di astrazione crescenti.

Kernel linux

Ai piedi dello stack si trova il kernel Linux, il quale permette di accedere all'hardware sottostante. All'interno del kernel sono collocati i driver per il funzionamento dell'interfaccia Wi-Fi, del display, dell'alimentazione etc... È inoltre presente anche un meccanismo di Inter Process Communication, che serve a far comunicare diverse componenti in un ambiente dove ogni applicazione diversa viene eseguita in un proprio processo. Il kernel Linux soddisfa il bisogno di avere un sistema operativo

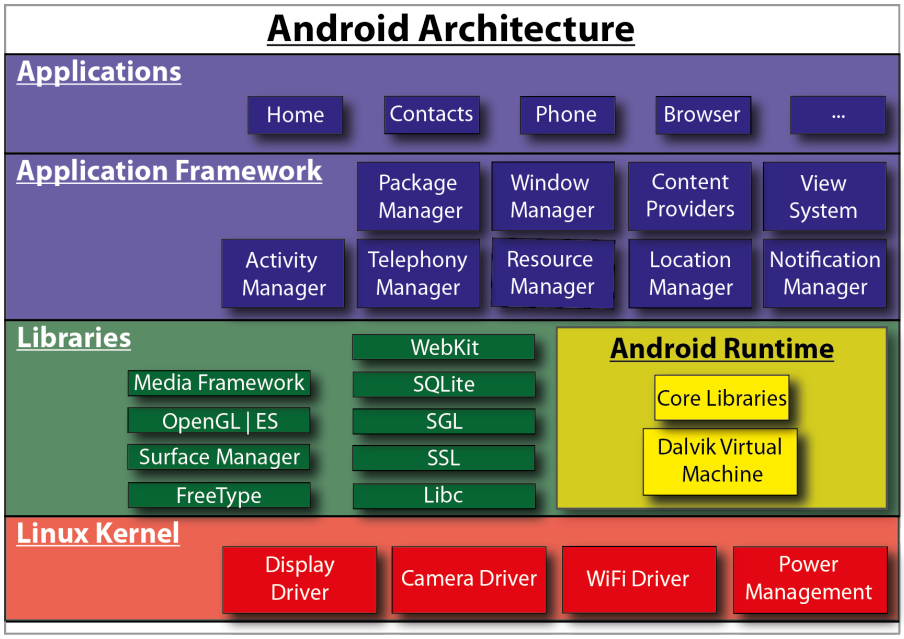

Figura 2.1: Schema dello stack

in grado di fornire meccanismi di sicurezza, di gestione della memoria, di gestione dei processi e che sia un sistema affidabile e aperto.

Core Libraries

È noto che, per sviluppare una applicazione in Java, non basta solamente l'applicazione in sè, ma servono anche le classi relative all'ambiente in cui viene eseguita. Mentre per la J2SE servono le classi relative ai package java e javax contenute nel file rt.java, per le applicazioni Android vale lo stesso discorso, ma con la differenza che in fase di compilazione si ha bisogno del file android.jar per creare il byte-code Java, mentre, durante l'esecuzione, il dispositivo metterá a disposizione la versione dex del runtime, la quale costituisce appunto la core library.

Dalvik

È la macchina virtuale usata per eseguire tutte le applicazioni Android. Questa è in grado di garantire che le applicazioni vengano eseguite nel migliore dei modi dal punto di vista della performance e dei consumi, in quanto è una macchina virtuale ottimizzata per l'esecuzione di applicazioni su dispositivi embedded e quindi con risorse hardware limi-

tate. Su Dalvik gira un codice ottenuto dal bytecode Java e questa scelta consente di avere tra l'altro una ridotta dimensione delle applicazioni Android, che puó arrivare fino al 50%. A differenza di altre virtual machine questa non è implementata a stack, ma a registri; infatti, una virtual machine implementata a stack presenta il grosso svantaggio di richiedere piú istruzioni per prelevare dati dallo stack stesso. In una macchina virtuale implementata a registri si ha il vantaggio che i registri di origine e destinazione sono codificati all'interno delle istruzioni, rendendole quindi piú lunghe ma in numero minore.

Libraries

Appena sopra il kernel si trova un componente che contiene varie librerie native, le quali rappresentano il core vero e proprio di Android. Tra queste:

- **Surface Manager:** si occupa di gestire ció che deve essere visualizzato in un determinato momento sullo schermo. Deve organizzare e scrivere tutte le finestre in maniera opportuna all'interno di un buffer.
- **Open GL ES:** insieme di librerie che forniscono l'accesso a funzionalitá 2D e 3D in dispositivi embedded.
- **SGL:** costituisce il motore grafico per Android.
- **Media Framework:** gestisce i codec per i formati di acquisizione e riproduzione audio e video.
- **FreeType:** motore per il rendering per i font.
- **SQLite:** utile per sviluppare DB relazionali.
- **WebKit:** browser engine basato su tecnologie HTML, CSS, Javascript e DOM.
- **Secure Socket Layer:** per la sicurezza nelle comunicazioni TCP-/IP.
- **Libc:** implementazione della libreria standard C libc, ottimizzata per i dispositivi embedded basati su Linux.

Application Framework

Componenti di alto livello che utilizzano le librerie appena esposte. Sono elementi utilizzabili dalle applicazioni che vengono sviluppate; ogni elemento permette di utilizzare un particolare insieme di funzionalitá. Tra i piú importanti si trovano:

- **Activity Manager:** organizza le varie schermate di un'applicazione all'interno di uno stack a seconda dell'ordine con il quale sono eseguite.
- **Package Manager:** si occupa del ciclo di vita dell'applicazione.
- **Content Provider:** si occupa della condivisione di informazioni tra processi.
- **Resource Manager:** mette a disposizione delle API per la gestione delle risorse associate alle applicazioni.
- **Telephony Manager:** consente di interagire con le funzionalitá caratteristiche di un telefono. In questo lavoro di tesi si è fatto uso di questa componente per rilevare l'intensitá del parametro RSSI.
- **Location Manager:** utilizzabile per le applicazioni Location Based. Anche in questo caso, per questo lavoro di tesi, si è fatto uso di questa componente per associare, ad un determinato valore del parametro RSSI, una coppia latitudine-longitudine.

Applications

Sullo strato piú alto dello stack si trovano tutte le applicazioni native e non native che utilizzano gli strati sottostanti durante il loro ciclo di vita.

2.1.3 Ciclo di vita ed elementi di una applicazione

In questa sezione si descrivono gli elementi costitutivi di una applicazione Android. Tali elementi sono descritti all'interno di un file XML di fondamentale importanza: il *Manifest File*. Verranno descritte con maggiore dettaglio le componenti che sono state impiegate per lo sviluppo dell'applicazione mobile CSAMS.

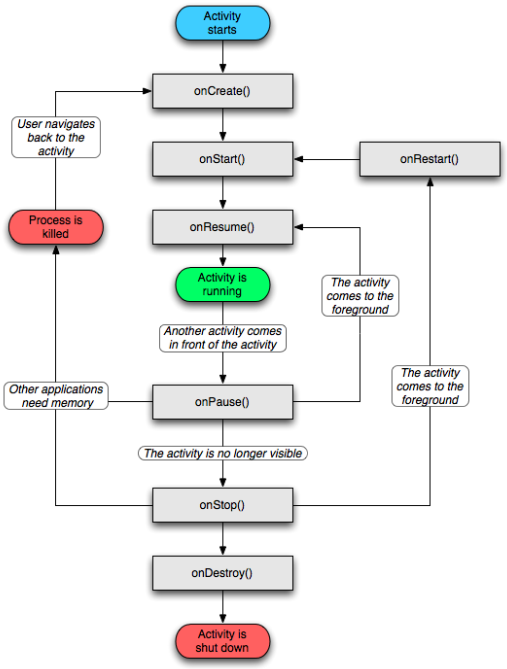

Figura 2.2: Ciclo di vita di una Activity

Activity

Questa componente puó essere immaginata come una schermata visualizzata sul display del terminale. Per creare una activity è necessario scrivere una classe che estenda la classe Activity. È possibile mostrare a video una Activity per volta e in questo caso si parla di Activity in foreground; le altre invece sono mantenute in background e quindi non sono in quel momento visualizzate. Le Activities sono infatti impilate una sopra l'altra e con il tasto BACK del terminale è possibile passare da una schermata alla precedente. Ogni applicazione gira all'interno di un unico processo di sistema, quindi ogni Activity condivide quest'ultimo. Queste componenti hanno un preciso ciclo di vita che non è controllato dalle applicazioni Android, ma piuttosto viene controllato dall'ambiente di esecuzione. Lo stato di una applicazione è determinato in base alla sua posizione all'interno dell'Activity Stack: quando una schermata viene visualizzata sul display allora vuol dire che si trova in cima allo stack. Quando viene premuto il tasto BACK l'Activity corrente diviene inattiva e viene prelevata dallo stack quella immediatamente precedente. Ogni passaggio di stato ha associato un handler, ovvero una funzione nella quale è possibile inserire delle operazioni da fare necessariamente

prima di passare allo stato successivo. Per non rischiare l'esaurimento delle risorse messe a disposizione dal sistema, nell'ambiente Android è presente un gestore della memoria: questo si occupa di esaminare lo stack e di decidere, in base alle prioritá delle Activiy presenti, quale applicazione terminare per liberare una parte di memoria e di creare quindi nuovo spazio libero al momento dell'apertura di nuove applicazioni. Il ciclo di vita di una Activity (Vedi Figura 2.2) è composto da diversi stati:

- **Active:** in questo stato, l'Activity si trova in cima allo stack e viene mostrata a video la rispettiva schermata. Quando una Activity diventa attiva è probabile che il gestore della memoria ne abbia rimosse altre nelle ultime posizioni dello stack. Da questo stato si puó uscire nel momento in cui si assiste all'entrata di una nuova Activity: in questo caso si dice che la precedente passa nello stato di Paused.

- **Paused:** quando una Activity passa in questo stato puó essere considerata in qualche modo ancora attiva ma con la differenza che non è piú concessa l'interazione con l'utente. Una Activity in stato di Paused è una possibile candidata nella terminazione da parte del gestore della memoria.

- **Stopped:** in questo caso l'Activity non è piú visibile e si mantengono in memoria solo le informazioni sul suo stato; come in precedenza, anche una Activity in stato di Stopped puó essere scelta per essere eliminata in modo da liberare memoria.

- **Inactive:** quando una Activity passa in questo stato, è stata terminata dal sistema da un precedente stato di Paused o Stopped; deve essere pertanto riavviata per entrare nuovamente nello stack. In questo caso, una applicazione che ha molte Activities nello stato Inactive, ha una maggiore probabilitá di essere terminata.

Service

Come le Activities, anche i Services hanno un loro ciclo di vita (vedi Figura2.2) ed eseguono varie operazioni; a differenza delle altre, questi eseguono le proprie elaborazioni in background, utilizzando sempre il thread principale dell'applicazione. Un Service viene avviato quando si devono effettuare delle operazioni costose dal punto di vista temporale; solitamente, quando viene creato, un Service crea a sua volta uno o piu thread che effettuano le varie operazioni. Questo meccanismo garantisce

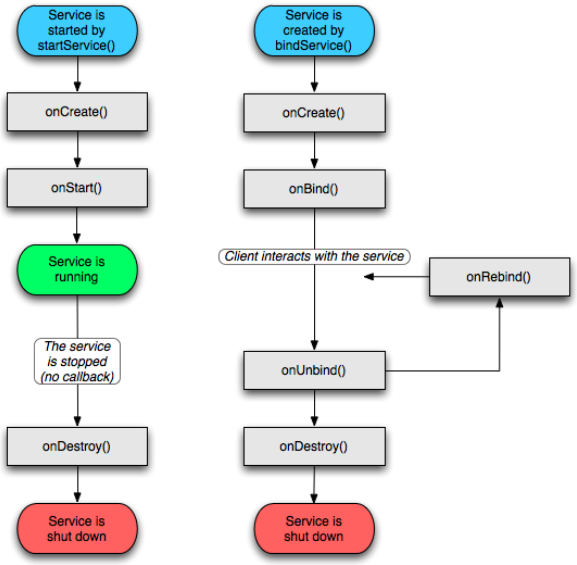

Figura 2.3: Ciclo di vita di un Service

lo svolgimento di tutte le operazioni dell'applicazione anche se questa possiede Activity non visibili. Si evita inoltre il congelamento dell'interfaccia grafica: in questo modo l'esperienza dell'utente rimane sempre positiva e non si nota nessun rallentamento delle prestazioni. Dalla descrizione del loro comportamento, si puó facilmente intuire che i Service, a differenza delle Ativities, non hanno una propria schermata da visualizzare e quindi non richiedono nessuna interazione con l'utente del dispositivo. Il ciclo di vita dei Services differisce da quello delle Activity, in quanto i primi hanno solamente due stati possibili:

- **Active:** i Services possono essere utilizzati da qualunque applicazione presente sul dispositivo; è importante quindi che il Service in questione si sia reso disponibile a ricevere determinate richieste. La terminazione di un Service puó essere causata da un messaggio di terminazione da parte di chi lo sta utilizzando, oppure dall'ambiente di esecuzione per necessitá di memoria. I Services possono fare uso di opportuni meccanismi di callback per comunicare con altri servizi che stanno in back-end oppure con Activities in front-end.

- **Stopped:** il Service è stato interrotto a causa di un messaggio di terminazione o dall'ambiente di sistema.

Intent

Un Intent puó essere pensato come la descrizione astratta di un operazione da eseguire: in altre parole si tratta di una intenzione di voler effettuare una particolare azione. Questa azione puó consistere nel lancio di una nuova Activity, oppure di un nuovo Service etc... . Questo strumento puó essere utilizzato per segnalare particolari eventi lanciati da una applicazione in esecuzione; le varie componenti di altre applicazioni possono registrarsi a questi particolari eventi in modo da ricevere delle notifiche ogni volta che si verificano. Un Intent è in grado di trasportare dati utilizzabili dal componente che lo riceve; tali informazioni vengono inserite dal componente che lo sta creando. Gli Intent si possono classificare in due modi:

- **Espliciti:** quando l'intent viene creato specificando il nome dell'Activity o del Service da eseguire. Questo tipo di Intent viene usato per scambiare messaggi e eventi all'interno di una solita applicazione in quanto i nomi delle varie componenti sono generalmente a conoscenza dello sviluppatore dell'applicazione.

- **Impliciti:** quando non si specifica il nome ma una particolare azione che Services o Activities devono eseguire. Se è presente nel sistema un componente capace di compiere tale azione, ricevendo l'Intent implicito, viene immediatamente attivato.

Grazie al lancio degli Intents, Android diventa una piattaforma nella quale i componenti al suo interno sono indipendenti ma situati in un ambiente completamente interconnesso. Le informazioni necessarie per descrivere un Intent sono due: *action* e *data*. Il primo descrive l'azione che deve essere svolta, mentre il secondo descrive il dato sul quale operare l'azione. Specificando questa coppia, piú ricevitori potrebbero essere in grado di accogliere l'Intent; per questo motivo viene coinvolto il *decision process*. Questo è un processo svolto da Android e si occupa di rintracciare tutti i possibili ricevitori di un Intent. Oltre a questi attributi essenziali ce ne sono altri che possono essere inclusi in un Intent:

- **category:** fornisce ulteriori informazioni sull'azione che si deve compiere.

- **MIME type:** specifica il tipo della risorsa coinvolta dall'Intent se non si conosce il valore del parametro data. Viene usato dal decision process per eleggere tra i potenziali ricevitori quelli che possono essere in grado di supportare questo tipo.

- **component:** in questo campo puó essere specificato il nome dell'Activity o del Service al quale è destinato l'intent. Se presente, rende inutile la coppia <action, data> in quanto le operazioni da compiere sono implicite nelle azioni che il ricevente eseguirá al suo avvio; in questo caso si parla di routing esplicito (il routing implicito si ha quando piú componenti possono ricevere l'Intent in questione).

- **extras:** contiene informazioni di vario tipo che si desidera inviare al ricevitore.

Un componente Android puó supportare solo un set di valori possibili per i parametri citati se esplicitamente scritto sul manifest file dell'applicazione.

Per quanto riguarda l'instradamento degli Intents si ha una terza categoria: i Broadcast Intents. In questa tipologia, tutti i parametri sopra citati diventano facoltativi e gli Intents di questo tipo vanno "in pasto" a tutti i ricevitori identificati come una nuova entitá: il Broadcast Receiver.

Broadcast Receiver

Questi componenti sono i ricevitori dei Broadcast Intents. Per rendere disponibile un componente di questo tipo è necessario crearlo e registrarlo. Per quanto riguarda la registrazione, puó essere fatta in due modi diversi:

- **registrazione statica:** questo tipo di registrazione viene effettuata andando a inserire esplicitamente all'interno del manifest file dell'applicazione il broadcast da registrare. Un Broadcast Receiver registrato in questo modo è in grado di ricevere Intents a lui destinati anche se l'applicazione in quel momento non è in esecuzione: nel momento della ricezione l'applicazione viene automaticamente attivata rendendo cosi Android un sistema event-driven.

- **registrazione dinamica:** il Broadcast Receiver viene registrato grazie all'esecuzione di particolari istruzioni a tempo di esecuzione. In questo caso, se l'applicazione non si trova in esecuzione al momento del lancio dell'Intent, questa non sará in grado di riceverlo.

Sia nel caso della registrazione statica che nel caso della registrazione dimanica, i Broadcast Receiver hanno bisogno che sia specificato un filtro

che elenchi le *actions* degli Intents che si desiderano intercettare. Ogni Broadcast Receiver ha un proprio handler che contiene l'insieme delle operazioni che si desiderano effettuare al momento della ricezione dell'Intent di tipo broadcast.

2.2 SWE

In questa sezione verrá descritto il framework SWE (Sensor Web Enablement), utilizzato per implementare parte del sistema lato server di questo lavoro di tesi. Tale framework viene messo gratuitamente a disposizione dal consorzio 52North: una rete no-profit di partner presenti nel campo della ricerca, dell'industria e della pubblica amministrazione. Grazie a questo framework si ha la possibilitá di integrare una rete di sensori di diverso tipo all'interno di una infrastruttura di rete. Con SWE, la 52north porta avanti un processo di standardizzazione iniziato dall'Open Geospatial Consortium (OGC): un altro consorzio che si occupa di standardizzare la gestione delle reti di sensori. In figura 2.4, si nota come SWE funga da intermediario tra la rete di sensori e il mondo esterno, il quale richiede informazioni provenienti dai sensori stessi.

2.2.1 Architettura di SWE

Grazie alla framework SWE, è possibile realizzare un sistema costituito da componenti anche molto diverse tra loro dal punto di vista dell'hardware; quindi possiamo considerarla di fatto come un middleware che presenta alle applicazioni clienti un unica rete service-oriented, nascondendo l'eterogeneitá della rete stessa. Il lavoro offerto dalla 52North si occupa di fornire alcuni standard per quanto riguarda il processo di discovery di istanze dei vari sensori registrati nella rete, lo scambio delle osservazioni prodotte dai vari dispositivi sensori, l'elaborazione di tali osservazioni e i messaggi scambiati tra sistema e sensori per permettere di iniziare una nuova campagna di acquisizione di dati ambientali quando necessario (processo di tasking). Le funzionalitá presenti in una generica rete di sensori sono state implementate da SWE stabilendo standard e interfacce:

- **Observation & Measurements Schema (O&M):** grazie a questa componente è possibile utilizzare particolari schemi XML standard in modo da codificare tutte le osservazioni.

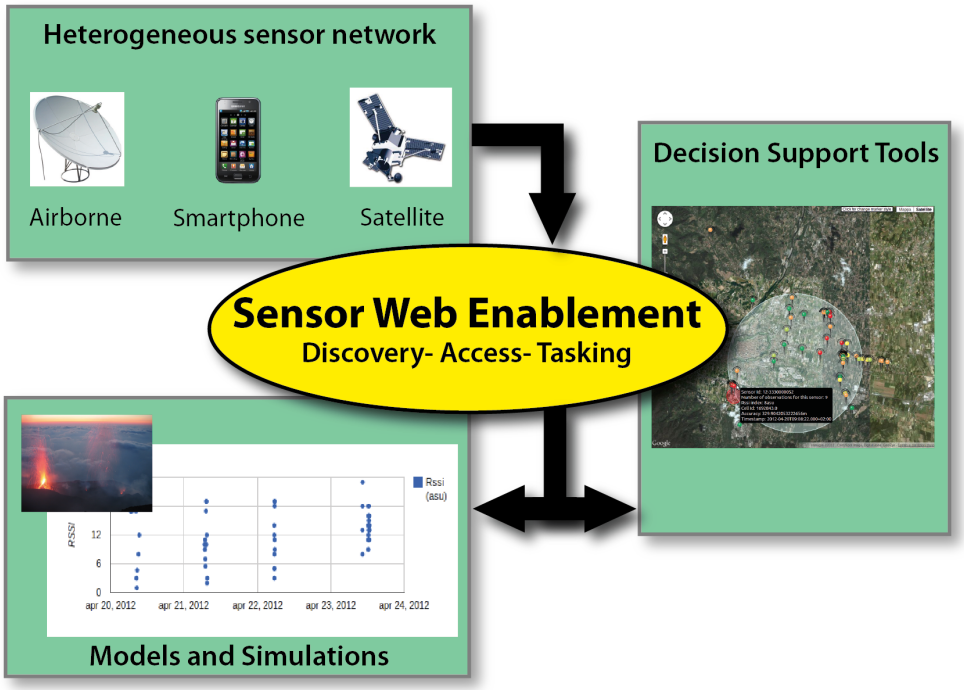

Figura 2.4: Sensor Web Enablement

- **Sensor Model Language (SensorML):** anche in questo caso vengono specificati alcuni modelli e schemi standard XML per la descrizione dei vari sensori.
- **Transducer Markup Language (TransducerML o TML):** vale lo stesso discorso fatto al punto precedente, ma con la differenza che in questi schemi XML si vogliono descrivere i trasduttori, i dati da loro prodotti e i fenomeni misurati.
- **Sensor Observation Service (SOS):** questa componente, di fondamentale importanza, rende possibile l'inserimento nel database, il recupero dal database stesso e il filtraggio delle osservazioni prodotte in una certa rete di sensori. Si pone nel mezzo tra un database che memorizza i vari dati e una eventuale applicazione cliente che sta cercando di recuperarli.
- **Sensor Planning Service (SPS):** anche questa componente è molto importante ai fini del corretto funzionamento dell'intero sistema; infatti costituisce l'insieme delle operazioni che servono a gestire la rete. È grazie a questa componente che si possono in-

iziare nuove capagne di task secondo vari parametri descritti nelle singole richieste. Si pone nel mezzo tra l'ambiente di gestione dei sensori e una eventuale applicazione cliente.

- **Sensor Alert Service (SAS):** questa framework realizza un sistema di publish-subscribe che comunica alle applicazioni clienti registrate a questo servizio di produrre di nuove osservazioni.

- **Web Notification Service (WNS):** permette l'invio asincrono di messaggi da parte di SAS e SPS.

Grazie a tutte le componenti sopra citate è possibile ottenere una rete di sensori completamente integrabile e gestibile attraverso la rete Internet; lo scopo di queste è infatti nascondere alle applicazioni clienti del servizio l'eterogeneitá presente tra queste due tipologie di reti. Per questo motivo, è di fondamentale importanza la presenza di una connessione dati in tutti i dispositivi sensori per renderli disponibili in remoto.

Verrano adesso descritti nel dettaglio i framework che sono stati ampiamente utilizzati dall'applicazione cliente RSSIWatcher: SOS e SPS. Tali componenti sono stati installati su un server web e sono quindi accessibili con il protocollo HTTP.

2.2.2 SOS

Sensor Observation Service è un servizio che mette a disposizione delle API da utilizzare quando si vuole memorizzare, recuperare e organizzare dati e metadati relativi ad un sistema di sensori. A prescindere dal fatto che i sensori siano statici (fermi in un punto preciso del territorio) o dinamici (liberi di muoversi), le misurazioni da loro prodotte vengono recuperate in maniera analoga e senza nessuna distinzione. Questo è un compito molto importante, in quanto storicamente gli utenti erano divisi in maniera netta tra utenti di sensori statici e utenti di sensori dinamici; ognuno di loro aveva proprie prospettive ed esigenze. L'eterogeneitá dei sensori viene nascosta grazie al concetto di *costellazione*: un insieme di sensori di vario tipo indirizzati dall'esterno come un entitá unitaria. Ogni costellazione è accessibile tramite il framework SOS. Le API messe a disposizione dell'utente per utilizzare i singoli sensori si compongono di diverse operazioni, le quali possono essere divise in *core operations* (per il funzionamento basilare del framework) e *transactional operations* (per le funzionalitá avanzate). Di seguito vengono introdotti alcuni termini utilizzati nell'ambito di SOS:

Osservazione

Con questo termine si intende il dato ottenuto dopo un processo di misurazione da parte di un sensore. Questo dato è codificato come una n-upla di valori numerici: ad esempio, un dato prodotto da un sensore GPS, è composto dalla coppia di valori latitudine-longitudine, mentre quella prodotta da un accelerometro, è composta dai tre valori numerici che rappresentano il valore dell'accelerazione nei tre assi cartesiani di riferimento. Un osservazione è classificata in base ai seguenti campi:

- **eventTime:** periodo temporale per il quale è stata ottenuta l'osservazione.
- **featureOfInterest:** oggetto su cui è stata fatta la misurazione (ad esempio un area geografica).
- **observedProperty:** fenomeno misurato (come ad esempio una localizzazione geografica o l'analisi dell'accelerometro).
- **procedure:** sensore che ha fornito tale osservazione.

Observation Offering

Con questo termine si identifica un raggruppamento di osservazioni, rese disponibili dal servizio offerto da SOS, che sono in relazione tra loro. Questo raggruppamento logico deve avere la caratteristica di essere *denso*: una query che specifica i parametri caratteristici di una offering non deve quindi mai presentare come risultato un insieme vuoto.

Sensor Data Consumer

Un entitá di questo tipo puó essere identificata da una applicazione che ha l'interesse di recuperare i dati messi a disposizione dai sensori, ai fini di espletare determinati servizi. Puó accadere che una applicazione di questo tipo non conosca a priori l'esistenza di eventuali istanze di SOS alle quali rivolgere le proprie richieste. A questo problema si puó ovviare con un processo di *discovery* delle istanze di SOS presenti in rete andando a interrogare i cosiddetti *Catalog Services* (CS): servizi che permettono alle istanze di SOS di registrarsi, in modo da rendersi visibili in rete nei confronti dei Sensor Data Consumer. Un CS, se interrogato, fornisce un elenco di istanze di SOS capaci di soddisfare i requisiti presentati da un SDC nella propria richiesta (vedi Figura 2.5 per il diagramma completo). A questo punto, un SDC puó richiedere le osservazioni che

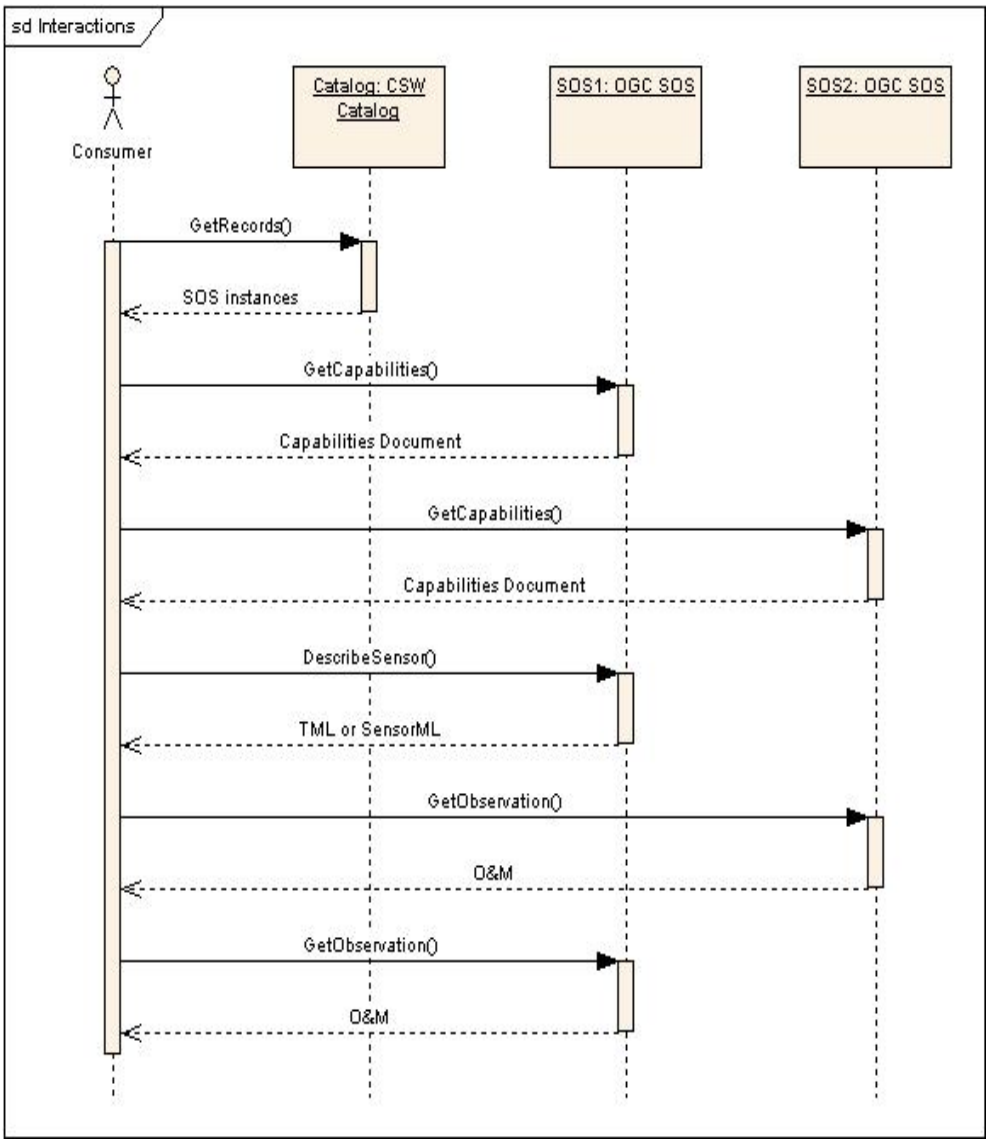

Figura 2.5: Diagramma temporale delle azioni di un SDC

desidera, oppure puó richiedere informazioni sulle offering o sui sensori associati alle offering.

Ci sono tre operazioni fondamentali appartenenti alla categoria delle core operations, messe a disposizione di un SDC:

- **GetObservation:** questo metodo offre la possibilitá di recurerare, con una query verso il database di SOS, le osservazioni in esso contenute. I criteri che possono essere inseriti all'interno della query sono il periodo temporale di produzione dell'osservazione, il fenomeno osservato e l'identificatore del sensore che ha prodotto l'osservazione.

- **GetCapabilities:** utilizzando questo metodo una applicazione cliente che ha intenzione di utilizzare i servizi offerti da una certa istanza di SOS, puó prelevare i vari metadati che forniscono una descrizione di tale istanza. Ad esempio, grazie a questo metodo, l'applicazione cliente puó recuperare le offering o la lista dei sensori registrati al servizio etc...

- **DescribeSensor:** restituisce metadata codificati in linguaggio SensorML o TML per fornire una descrizione delle caratteristiche hardware della procedura e delle grandezze che è in grado di misurare.

Sensor Data Producer

Un Sensor Data Producer è l'applicazione che si serve delle operazioni messe a disposizione dal profilo *transactional* di SOS. Tale profilo permette ai sensori di registrarsi a una istanza di SOS e di memorizzare le informazioni prelevate dall'ambiente, all'interno del database di competenza. Come per i SDC, anche i Sensor Data Producer possono non essere a conoscenza dell'esistenza di alcune istanze di SOS; anche in questo caso il problema viene risolto con l'interrogazione di un Catalog Service. Un alternativa, potrebbe essere invece specificare manualmente nel producer il percorso per arrivare a un istanza di SOS nota. Un sensore, prima di inviare le proprie osservazioni, deve effettuare un operazione di registrazione presso l'istanza di un SOS; in questo modo puó far conoscere a tale istanza le offering che è in grado di fornire. Solo a questo punto un sensore puó mandare i dati verso il lato server. Per ottimizzare questo processo, un SDP potrebbe effettuare preventivamente una richiesta di GetCapabilities, in modo da esaminare le offering giá presenti in SOS e, nel caso in cui una o piú offering supportate dal sensore siano giá presenti

nell'elenco, puó fare a meno di effettuare la registrazione completa. Ecco nel dettaglio le operazioni messe a disposizione del SDP dal transactional profile (vedi Figura 2.6):

- **RegisterSensor:** è l'operazione che consente ai SDP di effettuare il processo di registrazione presso SOS. Il sensore che la utilizza, al termine della registrazione, riceve un identificatore univoco all'interno dell'SOS. Tale identificativo sará inserito nell'elenco delle procedure gestite da tale SOS. La registrazione preventiva presso un'istanza di SOS è una condizione necessaria, da parte del sensore, affinchè esso possa mandare le osservazioni da lui prodotte verso il lato server.

- **InsertObservation:** questa operazione consente a un SDP di poter inserire, all'interno del database, le nuove osservazioni prodotte in modo da renderle disponibili alle applicazioni clienti che le richiedono. L'osservazione da inserire deve essere codificata secondo le specifiche O&M all'interno di un file XML e deve contenere tra l'altro anche l'id ricevuto al termine della fase di registrazione. È quindi necessario che il sensore abbia preventivamente memorizzato tale identificativo per poterlo riproporre in questa richiesta.

Dopo aver descritto tutte queste componenti, è facile immaginare le motivazioni che hanno spinto a scegliere il modulo SOS per realizzare parte del lato server in questo lavoro di tesi: disaccoppiando totalmente SDP e SDC, uno non deve essere per forza a conoscenza della presenza dell'altro e inoltre, il fatto che si seguano gli standard definiti in SWE porta alla conseguenza che tutti i tipi di sensori possono essere trattati allo stesso modo, dai satelliti agli smartphone.

2.2.3 SPS

L'acronimo SPS significa Sensor Planning Service. Questo framework permette di gestire i sensori in esso registrati; grazie a lui è possibile infatti eseguire operazioni di rilevazione di dati ambientali in maniera analoga per tutti i vari tipi di sensori, anche se essi presentano tra loro grosse differenze in termini di hardware e software. SPS è sempre presente ogni volta che si ha la necessitá di avviare dei task ex-novo verso i sensori registrati, inoltrando loro le varie richieste. Per questo motivo SPS puó essere definito come un middlware che mette a disposizione di chi lo usa delle API da utilizzare nel momento in cui si desidera interagire con i sensori (vedi Figura 2.7 per il diagramma temporale delle operazioni).

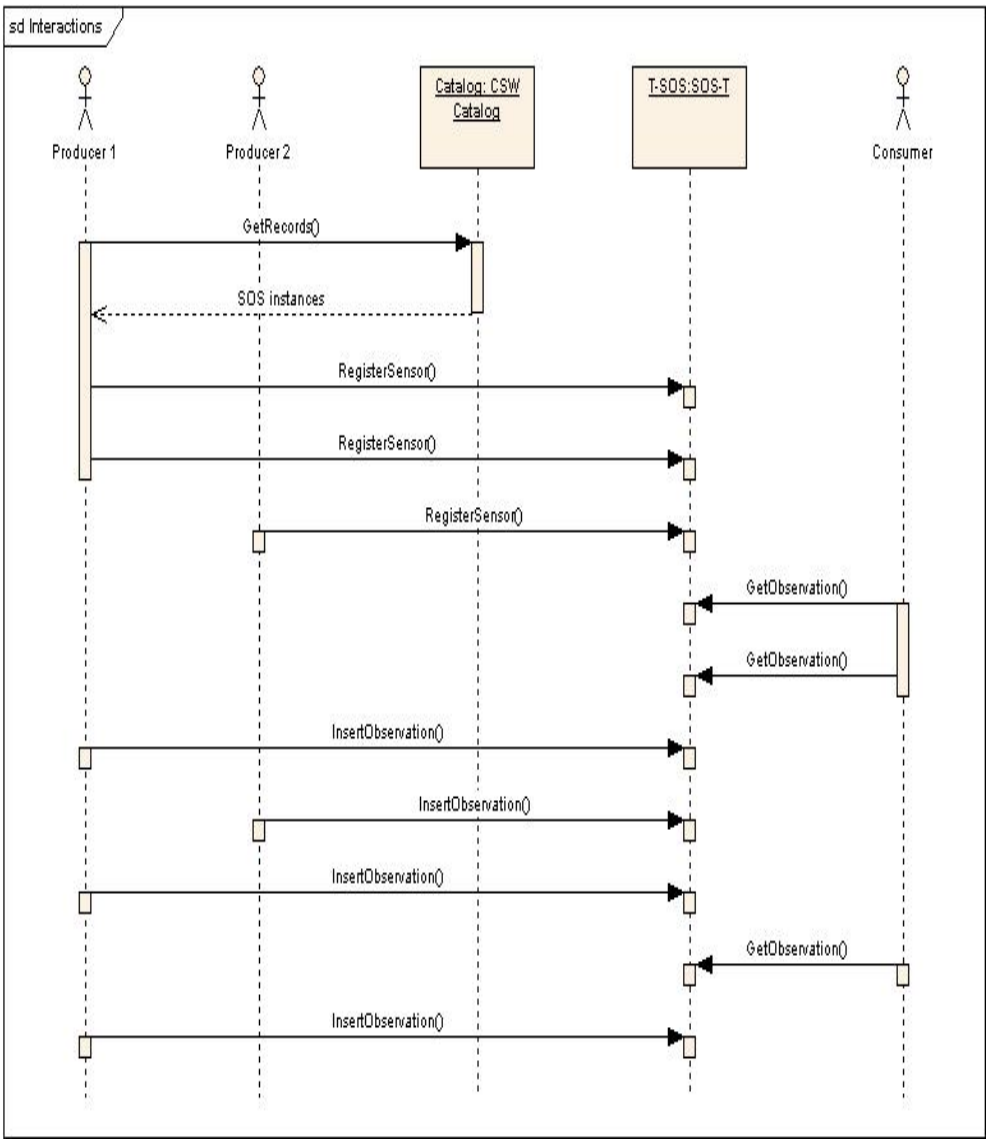

Figura 2.6: Diagramma temporale delle azioni di un SDP

Le operazioni messe a disposizione dalla framework si dividono in due categorie:

Informational operations

Sono operazioni che vengono utilizzate per recuperare informazioni sulle caratteristiche sia dell'istanza SPS, che dei sensori ad essa registrati. Tra queste operazioni si individuano in particolare:

- **GetCapabilities:** serve a permettere, a una applicazione cliente, di recuperare informazioni riguardanti quella particolare istanza di SPS. Il risultato di tale richiesta è un documento XML contenente tra l'altro la lista dei sensori registrati e i rispettivi fenomeni misurati.

- **DescribeTasking:** quando una applicazione cliente ha intenzione di utilizzare un particolare sensore, puó utilizzare questa operazione per recuperare informazioni relative ai parametri che dovranno essere inseriti nella richiesta di Submit. Verranno restituiti quindi un elenco di nomi con i relativi valori ammessi per quel particolare tipo di sensore.

- **DescribeResultAccess:** con questa operazione, una applicazione cliente è in grado di recuperare il percorso necessario per raggiungere l'istanza SOS nella quale questo particolare tipo di sensore ha depositato le sue informazioni. Il risultato di tale operazione è quindi un indirizzo di un qualche servizio atto alla memorizzazione dei dati prodotti dal sensore.

- **GetStatus:** fornendo l'identificativo di un task ricevuto dopo l'operazione di submit, una applicazione cliente, con questa richiesta, puó ricevere informazioni circa lo stato in cui si trova quel determinato task. Il risultato di questa operazione è una descrizione della fase in cui si trova (per esempio il task potrebbe essere in esecuzione oppure terminato).

Functional Profile

Questo profilo contiene quindi tutte le operazioni necessarie per gestire i task. Come giá accennato, quando un nuovo task viene avviato, anch'esso riceve un identificativo univoco all'interno del sistema;

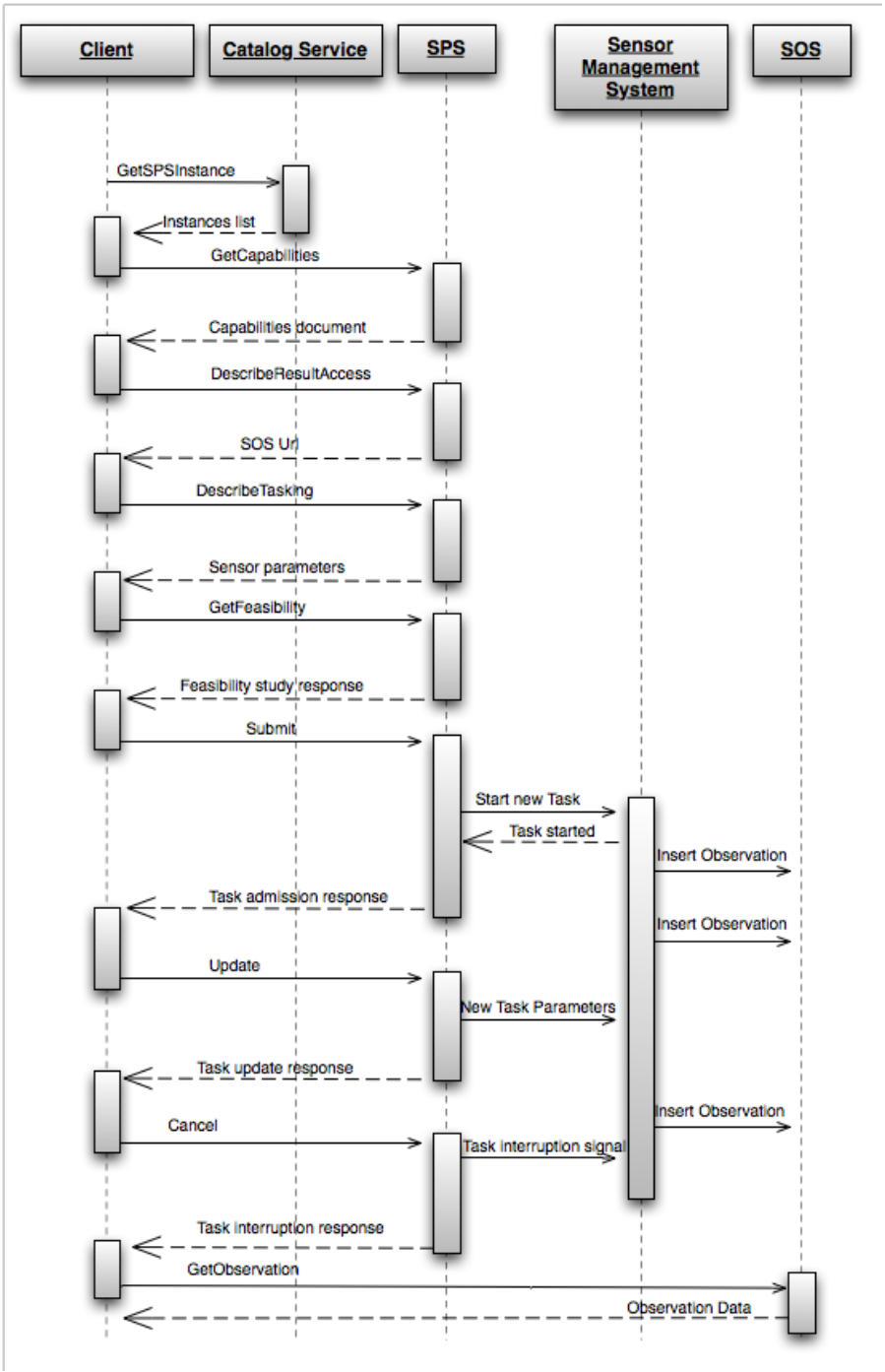

Figura 2.7: Diagramma temporale delle azioni di SPS

questo identificativo viene restituito all'applicazione cliente. In tal modo, l'identificativo ottenuto, puó essere sfruttato dal cliente dell'applicazione per conoscerne lo stato di avanzamento, oppure per chiederne la cancellazione. Le operazioni che fanno parte di tale profilo sono le seguenti:

- **GetFeasibility:** con questa operazione è possibile eseguire uno studio di fattibilitá su un particolare sensore. I parametri da inserire in questa richiesta, devono essere quelli che si intende inserire in una richiesta di Submit: in questo modo SPS sará in grado di prendere in considerazione tali parametri per capire se il sensore, in quel determinato istante, sia in grado di soddisfare correttamente la nuova richiesta di task. Il risultato è quindi un esito di accettazione o di rifuito. SPS è in grado di eseguire uno studio di fattibilitá andando ad accedere alle astrazioni in suo possesso dei singoli sensori: i *plugin* (di seguito verranno fornite informazioni su questa entitá). Lo studio di fattibilitá viene svolto in maniera diversa in dipendenza dal tipo di sensore che si sta prendendo in considerazione.

- **Submit:** operazione che consente di sottomettere nuovi task. Richiede particolari valori utilizzabili per impostare la nuova osservazione; tali parametri cambiano in base al particolare tipo di sensore che si intende utilizzare e vengono decisi dall'entitá che sta impostando la nuova richiesta. La risposta di questa operazione, in caso di esito positivo dell'accettazione del task, è un identificatore univoco: il *taskId*. Non è detto che il task entri subito in esecuzione e un cliente, qualora fosse interessato a scoprire se il task sia in esecuzione o meno, puó inoltrare una richiesta di GetStatus fornendo l'id ricevuto come risposta al Submit.

- **Update:** grazie a questo metodo un cliente puó effettuare una modifica ai parametri forniti in una richiesta di Submit precedentemente effettuata; dovranno essere forniti quindi l'identificatore univoco del task ottenuto con il submit e tutti i parametri necessari all'esecuzione del particolare task. In risposta, SPS restituisce un esito binario di accettazione o rifiuto dell'Update.

- **Cancel:** come nel caso precedente, un cliente che utilizza questo metodo deve fornire l'identificativo del task in questione. Il task identificato dal taskId inserito in tale richiesta, se attualmente in

esecuzione, verrá interrotto e verrá fornito al cliente un esito positivo. Nel caso in cui il task non possa essere interrotto oppure sia giá stato completato nel momento in cui arriva la richiesta di Cancel, il cliente riceverá un esito negativo.

2.3 CloudSensor

In questa sezione verranno fornite informazioni riguardanti il modulo che si occupa di rendere la rete di sensori disponibile e raggiungibile come un servizio web, svolgendo la funzione di middleware: il sistema *CloudSensor*. Questo sistema rende possibile la richiesta e il recupero dei dati ambientali prelevati dai sensori registrati presso un istanza di SOS, utilizzando il protocollo HTTP e documenti XML all'interno dei quali vengono inserite sia le descrizioni delle varie richieste che le conseguenti risposte. CloudSensor si configura come l'astrazione di una rete dinamica di sensori, i cui nodi sono appunto i vari sensori integrati all'interno degli smartphones. Nel seguito verranno descritte le due macrocomponenti che costituiscono il sistema CloudSensor: CloudSensor Service (CSS) che costituisce il front-end del sistema e CloudSensor Acquisition and Management System (CSAMS) che rappresenta l'applicazione mobile da installare sullo smartphone (vedi Figura 2.11 per avere una visione sull'architettura del sistema CS).

Questa componente si occupa di realizzare il servizio web che rende disponibili all'utente alcune delle operazioni messe a disposizione da SOS e da SPS. Ogni applicazione cliente del sistema deve pertanto sfruttare questo modulo, utilizzando il protocollo HTTP, in modo da effettuare le proprie richieste nei confronti della rete di sensori. In particolare CSS si occupa di: controllare la corretta sintassi delle richieste ricevute prima di inoltrarle a SPS e SOS, adattare le richieste di una applicazione cliente prima di inoltrarle ai terminali mobili in modo che essi possano comprenderla correttamente, inoltrare a CSAMS nuove richieste in arrivo e costruire le risposte alle suddette richieste. Le singole componenti che costituiscono il modulo CSS sono scritte nelle sottosezioni che seguono.

2.3.1 Naming Server

Come giá accennato in precedenza, per utilizzare il sistema nel suo insieme, è necessario che le varie parti utilizzino il protocollo HTTP; ci si trova quindi in una situazione dove assiste alla presenza presenza degli indirizzi IP. I terminali mobili, con il loro attuale IP, si mettono

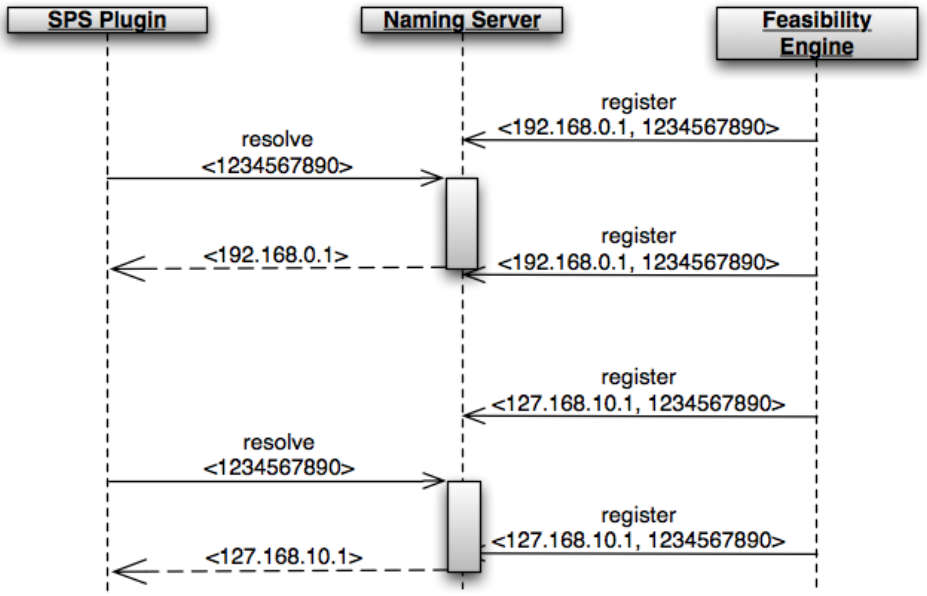

Figura 2.8: Esempio di interazione con il Naming Server

in attesa di nuove richieste su un numero di porta dedicato; è nota a tutti peró la caratteristica di volatilitá di tale indirizzo che quindi puó cambiare in ogni momento. Le applicazioni clienti, se vogliono sottomettere un nuovo task verso un nuovo sensore, devono essere in grado di raggiungerlo e quindi devono essere a conoscenza del suo attuale indirizzo IP. Per risolvere questo problema, c'è bisogno peró di essere a conoscenza di una informazione che sia statica nel tempo in ogni smartphone, in modo da associarla all'informazione dinamica dell'indirizzo IP; una possibile soluzione potrebbe essere il proprio numero telefonico. Associando il numero telefonico all'indirizzo IP attuale è possibile quindi costruire dei bind *<numero di telefono, indirizzo IP>* e recuperare l'informazione dinamica sull'IP andando a utilizzare come chiave di ricerca l'informazione statica costituita dal numero di telefono. Questi bind devono essere rinfrescati periodicamente da parte dello smartphone che, trovandosi in mobilitá, è spesso soggetto a cambiamenti di indirizzo IP.

Il Naming Server è quindi la componente che svolge le operazioni di traduzione da numero telefonico a indirizzo IP e che memorizza i bind, aggiornandoli nel momento in cui ne riceve di nuovi dai vari smartphones registrati. Si tratta di una Java Servlet che riceve quindi le sue richieste HTTP, di tipo GET, attraverso appositi URL e rende disponibili al

modulo SPS e agli smartphones le seguenti operazioni:

- **Register:** permette agli smartphones di inviare verso il Naming Server una nuova coppia <numero di telefono, indirizzo IP>. Il Naming Server, una volta ricevuto il bind, lo memorizza nella tabella di traduzione a una nuova posizione (se non esistono vecchi bind associati a quel numero telefonico), oppure aggiornando il bind precedentemente inviato da tale dispositivo. Richiede come parametri solamente il numero di telefono e l'attuale IP che si vuole registrare.

- **Resolve:** operazione messa a disposizione del modulo SPS che la utilizza per ricavare l'IP corrispondente a un certo numero telefonico, nel momento in cui si vuole iniziare un nuovo task. Richiede come parametro il numero telefonico di uno smartphone e restituisce, se esiste e se non è scaduto il timer di validitá, l'indirizzo IP associato.

- **Cancel:** questa operazione puó essere utilizzata dai dispositivi mobili che desiderano cancellare il bind dalla tabella di traduzione. Richiede come parametro il numero di telefono dello smartphone.

Una possibile successione di eventi puó essere la seguente (vedi Figura 2.8): quando una applicazione cliente richiama una operazione di Submit, si preleva dall'ID del sensore registrato presso l'SPS il numero telefonico (gli identificativi dei sensori sono del tipo *indice-sensore:numero-telefono*). L'identificativo del sensore viene poi dato "in pasto" alla richiesta di Resolve, ricevendo come risultato l'indirizzo IP attuale necessario per raggiungere il terminale mobile che contiene il sensore di interesse. A questo punto, conoscendo su quale porta si trova in ascolto l'applicazione mobile CSAMS, si hanno a disposizione tutte le informazioni necessarie per inoltrare la nuova richiesta. CSAMS, durante il suo ciclo di vita, dovrá spedire verso il Naming Server dei bind periodici per ovviare al problema della volatilitá dell'indirizzo IP. Gli aggiornamenti sul bind sono obbligatori in quanto, a ogni bind, viene associato un timer di validitá che, una volta scaduto, causa l'eliminazione di tale bind dalla tabella di traduzione.

2.3.2 SPSPlugin

Questa componente viene utilizzata dal modulo SPS per avere in memoria una astrazione di un determinato tipo di sensore. Anche se

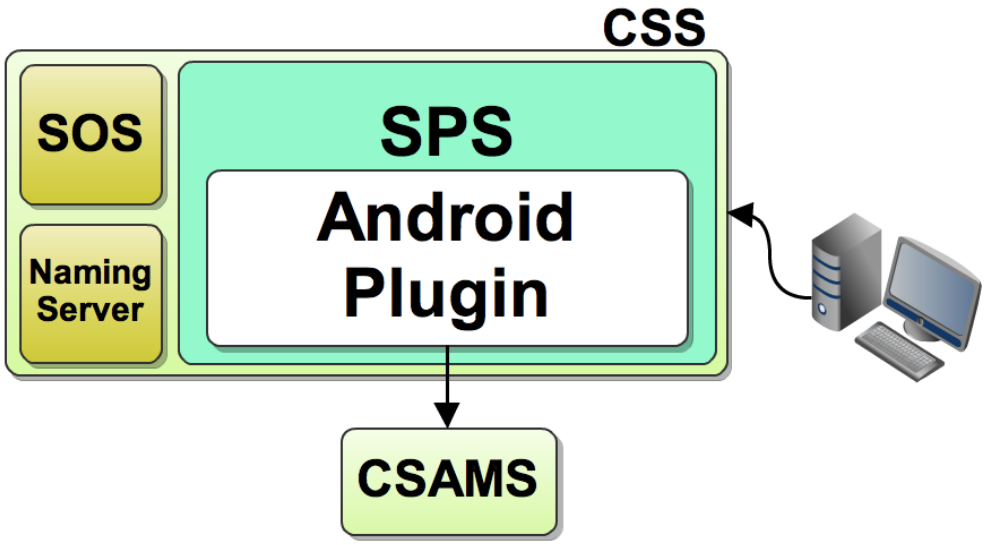

Figura 2.9: Android Plugin all'interno di CS

sui moderni smartphone si trovano diverse tipologie di sensori, questi possono essere caratterizzati da un interfaccia comune, in quanto ogni sensore viene considerato come una entitá che produce dati e che quindi puó trovarsi in stato attivo (se sta eseguendo un task) o nello stato disattivo (se non sta eseguendo alcun task). Per questo motivo, per astrarre un qualsiasi sensore presente in uno smartphone, SPS si serve di un solo plugin: l'*AndroidPlugin*. Quando un nuovo sensore tenta di registrarsi presso il modulo SPS, se l'operazione va a buon fine, viene creata una nuova istanza di tale plugin, la quale contiene le strutture dati e i metodi necessari per mantenere lo stato ed effettuare operazioni di Submit o di Cancel sul sensore stesso. Se si va verso un livello di dettaglio piú alto, all'interno dell'AndroidPlugin si trovano le seguenti componenti:

- L'URL relativo al Naming Server al quale si dovrá rivolgere ogni volta che dovrá inoltrare una richiesta di Submit verso un particolare sensore. Viene memorizzato all'atto della registrazione del sensore in SPS.

- I parametri che caratterizzano il particolare tipo di task, in modo da riuscire ad inoltrarli al sensore in attesa di nuove richieste. Questi parametri vengono intercettati e memorizzati in una struttura dati ogni qualvolta si esegua una chiamata alla funzione di submit.

- ID del task attualmente in esecuzione sul sensore.

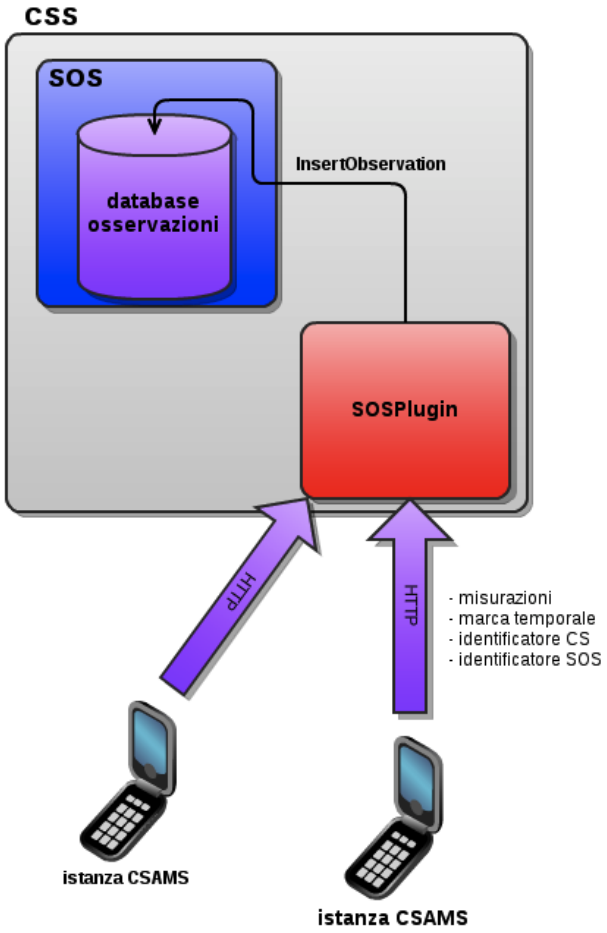

Figura 2.10: Plugin all'interno di CS

- Handler dell'operazione di Submit.

- Handler dell'operazione di Cancel.

Per creare un nuovo Plugin è sufficiente andare a derivare una particolare classe che si trova all'interno della framework offerta da 52North e ridefinire i particolari metodi che caratterizzano il comportamento del particolare tipo di sensore che si vuole creare. SPS richiamerá tali metodi quando vengono inoltrate le varie richieste. I parametri che descrivono i task sono recuperati dal documento XML passato durante la richiesta di Submit.

2.3.3 SOSPlugin

I sensori integrati negli smartphone, non fanno altro che prelevare un particolare tipo di dato ambientale. Questo dato è considerato "grezzo", in quanto su di esso non vengono effettuate alcune operazioni di raffinamento e viene quindi spedito verso il modulo SOS privo di modifiche; tale approccio porta al notevole vantaggio di risparmiare operazioni costose in termini di memoria e consumi da parte del terminale mobile, il quale ha come punto debole il tempo di vita della batteria. La componente di CSS che si occupa di ricevere questi dati grezzi e di inserirli correttamente in oppurtuni file XML da mandare a SOS, è proprio l'-SOSPlugin. Tali documenti XML vengono creati in base ai parametri che il sensore gli invia. Per il corretto funzionamento del sistema, non si richiedono solamente i dati prelevati dal sensore, ma anche una marca temporale, l'ID ricevuto in seguito alla registrazione al modulo SOS e l'identificatore del fenomeno misurato. Senza tutti questi dati, non sarebbe possibile inoltrare la richiesta di InsertObservation verso SOS. Ricapitolando, i parametri che l'SOSPlugin inserisce nel file XML sono i seguenti:

- identificatore SOS ricevuto in fase di registrazione del sensore che ha prodotto l'osservazione.
- identificatore del fenomeno ambientale misurato.
- marca temporale dell'osservazione.
- valori che compongono l'osservazione.

Le richieste che avanzeranno dai terminali mobili verso l'SOSPlugin saranno richieste HTTP di tipo GET; la stringa che identificherá tale richiesta sará del seguente tipo:

myhost:8080 / SOSPlugin?sensorID = X & phenomenon = Y & timestamp = Z & values = G,H,J

2.3.4 CSAMS

All'interno del sistema, la componente CSAMS è l'unica a risiedere nel lato "dispositivo mobile". Rappresenta quindi l'applicazione da installare sul terminale per gestire l'interazioni tra questo ed il lato server, il quale spedisce ad essa le varie richieste di task. L'applicazione mobile, una volta ricevuta la richiesta, si preoccupa di tradurla in chiamate ai driver

che gestiscono i sensori. Di seguito verranno descritte brevemente le parti che compongono l'applicazione CSAMS.

Interfaccia Utente

Grazie a una semplice interfaccia l'utente puó impostare alcune preferenze di funzionamento dell'applicazione andando a decidere, per esempio, un limite sulla frequenza di campionamento.

Boot Loader (BL)

Questa è la componente che si occupa di effettuare le giá citate procedure di registrazione verso i due moduli SPS e SOS. Oltre a questo importante compito, il Boot Loader si preoccupa di memorizzare i dati sulle preferenze di funzionamento impostate dall'utente tramite l'interfaccia grafica. Queste impostazioni possono riguardare i sensori che si desidera rendere disponibili, i limiti sulla frequenza di campionameto o gli URL delle componenti server con le quali CSAMS dovrá comunicare. Le registrazioni vengono effettuate grazie all'operazione di *RegisterSensor*, la quale ha bisogno di documenti SensorML o TML per descrivere i sensori che desidera registrare. Per evitare di registrare sensori giá registrati in precedenza, CSAMS consulta preventivamente un database contenente queste informazioni.

Feasibility Engine (FE)

Il Feasibility Engine è la prima componente che si accorge dell'arrivo di una richiesta di Submit e quindi occupa il livello piú alto nello stack dell'applicazione. Questa componente deve occuparsi di effettuare diverse operazioni:

- **Attendere richieste:** l'applicazione riceve richieste attraverso il protocollo HTTP, ponendosi in attesa su una apposita porta (8081 per default). Dopo aver ricevuto la richiesta di tipo GET, la codifica estraendo direttamente da essa i parametri del task.

- **Fare studio di fattibilitá:** una volta estratti i parametri relativi al nuovo task, questa componente si preoccupa di verificare se con questi sia possibile o meno effettuare una nuova osservazione. Puó accadere infatti che, per vari motivi, lo studio di fattibilitá ritorni con esito negativo. Per esempio, i parametri richiesti possono

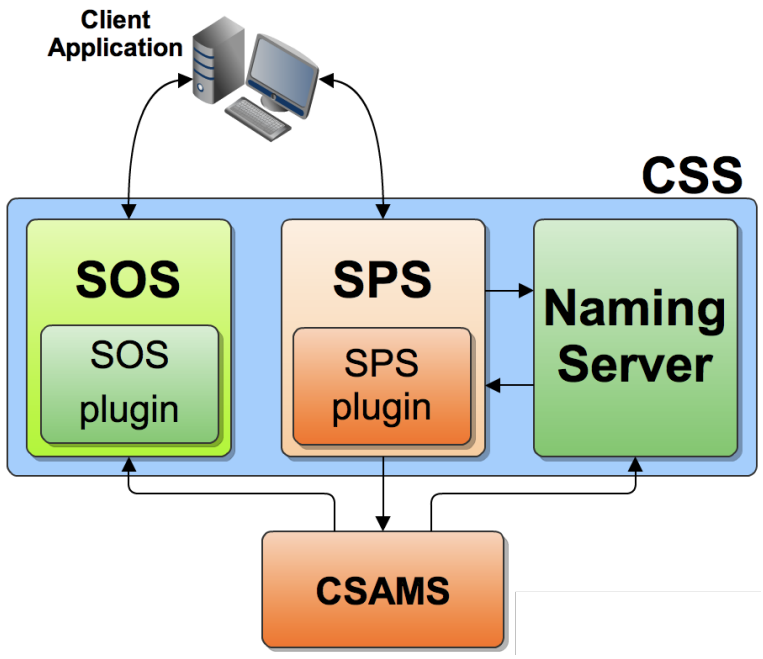

Figura 2.11: Architettura del sistema ad alto livello

andare oltre i vincoli imposti dall'hardware del dispositivo o dall'utente stesso, oppure il dispositivo potrebbe essere attualmente impegnato nell'esecuzione di un altro task. Se lo studio di fattibilitá viene superato con esito positivo, la richiesta viene passata al modulo successivo: il *Task Scheduler*.

- **Fare il binding con il Naming Server:** il FE si occupa tra l'altro di spedire periodicamente l' indirizzo IP attualmente assegnato al dispositivo, creando quelli che in precedenza sono stati indicati come *bind*. In questo modo, l'applicazione garantisce che il sensore sia reperibile in qualsiasi momento in quanto, il periodo con il quale i bind vengono spediti, deve essere ovviamente minore del periodo di validitá assegnato ai vari bind dal Naming Server stesso.

Task Scheduler (TS)

Riceve le richieste direttamente dal Feasibility Engine e si occupa di attivare o disattivare i sensori fisici. TS è quindi la componente che, a prescindere dal tipo di sensore coinvolto, avvia le componenti software che lo gestiscono. Grazie alla *task queue*, il TS riesce anche a tenere traccia dello stato di ogni sensore (attivo o disattivo) e restituisce questa

informazione al FE per consentirgli di effettuare correttamente lo studio di fattibilitá.

Wrapper

Questo elemento costituisce l'ultimo strato presente tra l'interfacciamento dei sensori veri e propri e il resto dello stack dell'applicazione CSAMS. Il Wrapper, una volta ricevuti i parametri dal TS, traduce il tutto in chiamate alle librerie messe a disposizione da Android e che gestiscono il particolare tipo di sensore coinvolto. All'interno di CSAMS esistono quindi tanti wrapper quanti sono i sensori che si vogliono gestire in CloudSensor; in questo lavoro di tesi si è rivelato necessario implementare quindi un Wrapper ex-novo, in quanto il sensore da utilizzare per prelevare il parametro RSSI non era supportato nella versione iniziale. Il Wrapper si occupa inoltre di associare la marca temporale ad ogni osservazione effettuata, l'ID del fenomeno misurato e l'ID del sensore prima di spedire il tutto al modulo *SOS Feeder*.

SOS Feeder (SF)

L'SOS Feeder è la componente che riceve dal Wrapper le osservazioni dopo che questo le ha arricchite con altre informazioni; dopo di che le spedisce al SOSPlugin, il quale a sua volta le inoltra verso il database gestito da SOS.

2.4 Servlets e Java Server Pages

In questa sezione viene descritta la tecnologia utilizzata per creare la web application RSSIWatcher. Con questa tecnica sono state inoltre realizzate varie componenti fondamentali del sistema: il modulo SOS, SPS, l'SOSPlugin e il Naming Server. Prima di proseguire, occorre fare una breve introduzione sullo strato software che accoglie le parti del sistema realizzate per mezzo delle Servlets e le JSP: il server Tomcat. Tomcat è un software open source e rappresenta un Web Application Server: un server capace di gestire e rendere disponibili varie applicazioni costruite con la tecnica delle Servlet o delle JSP. Tomcat, una volta avviato, si pone in ascolto su una porta (di default la 8080) e attende richieste nei confronti delle applicazioni che contiene. Tale Web Server mette a disposizione dell'utente una interfaccia per installare le varie web application; si parla in questo caso di processo di Deploy. Lo sviluppo di applicazioni

web based deve il suo successo a questa particolare tecnologia, introdotta per fornire ai programmatori un linguaggio lato server semplice da usare e compatibile con il linguaggio Java. Le servlet sono applicazioni Java installate in un ambiente di lavoro, detto container, che ne gestisce il funzionamento e il ciclo di vita ed hanno la possibilitá di sfruttare tutte le risorse presenti nel sistema in cui risiedono, avendo quindi a disposizione la possibilitá di cooperare per fornire all'utente finale servizi piú complessi. Esistono diverse tipologie di Servlet a seconda del protocollo di comunicazione utilizzato; per fornire informazioni compatibili con questo lavoro di tesi si prenderanno in considerazione le *HttpServlet*, presenti nel package *javax.servlet.http*. Dopo aver esteso tale classe è possibile creare Servlet in grado di comunicare con i client e di rispondere dinamicamente alle loro richieste. Lo scopo è quello di condividere un unico oggetto con tutti i clienti che ne richiedono i servizi; in questo modo vengono eliminate tutte le operazioni di creazione dell'oggetto, creazione di connessioni ed eliminazione al termine del servizio di connessioni ed oggetti creati. In questo modo vengono garantite velocitá elevate nell'espletamento dei vari servizi, anche se in un determinato momento il server si trova ad affrontare numerose connessioni; questo vantaggio si ottiene grazie al risparmio sulle risorse di sistema e grazie all' ottimizzazione della loro gestione. Tale approccio comporta peró anche aspetti negativi: ogni richiesta in arrivo al web server si traduce in una creazione di processi leggeri (thread) che concorrono, come giá detto, all'utilizzo delle stesse risorse. Una gestione inefficiente del multithreading potrebbe portare ad un possibile accesso inconsistente ai dati con un evidente errore dell'applicazione. Il metodo di funzionamento ricalca il paradigma client-server. La richiesta e la risposta sono rappresentate da due oggetti: *HttpServletRequest* e *HttpServletResponse* contenuti nel package *java.servlet.http*. L'oggetto HttpServletRequest rappresenta la richiesta del cliente che sta utilizzando il servizio offerto dalla servlet; è possibile quindi, grazie ad oppurtuni metodi offerti proprio da tale oggetto, recuperare eventuali parametri inseriti nella richiesta.

Le Java Server Pages rappresentano il livello di presentazione nella tecnologia J2EE. Queste non aggiungono quasi niente di nuovo rispetto alle normali Servlets. Le JSP sono infatti considerate un estensione delle Servlet Java e di conseguenza permettono di sfruttare tutte le funzionalitá di queste, aggiungendone di nuove. Le JSP sono dei file che contengono al loro interno sia linguaggio HTML che linguaggio Java, il quale consente l'utilizzo dinamico del servizio web. La separazione tra linguaggio Java e linguaggio HTML rappresenta la separazione tra il disegno del-

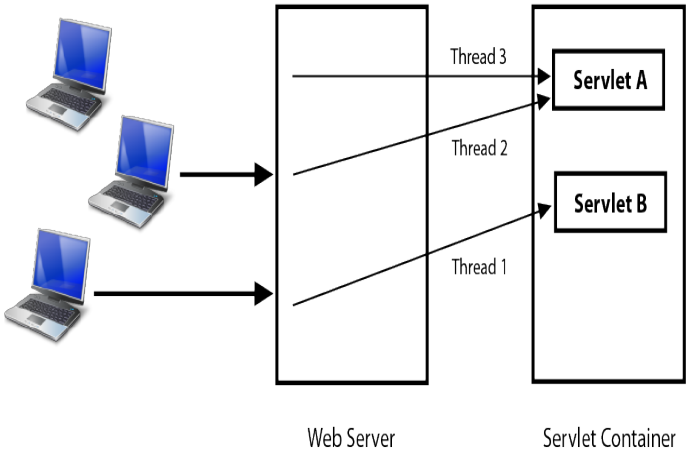

Figura 2.12: Ambiente di esecuzione delle Servlet

l'interfaccia grafica (o la presentazione dei risultati delle richieste) e la progettazione delle funzioni che costituiscono il servizio richiesto. Per effettuare questa separazione vengono utilizzati oppurtuni marcatori che indicano l'inizio e la fine di un pezzo di codice scritto in Java, ovvero il contenuto dinamico (quello che cambierá di volta in volta in base ai parametri di input inseriti nelle richieste dell'utente). Le JSP, alla fine della loro progettazione, vengono tradotte automaticamente in Servlet; si puó dire quindi che la differenza tra Servlet e JSP si traduce in una maggiore semplicitá da parte di quest'ultime nella gestione della presentazione dei risultati ottenuti. Il Naming Server, che non aveva necessitá di presentare visivamente dei risultati a un utente, è stato costruito con la tecnica delle classiche Servlets; mentre l'applicazione web RSSIWatcher è stata sviluppata con la tecnica delle JSP in quanto si aveva la necessitá di adattare alcuni contenuti grafici in maniera dinamica in base ai risultati di alcune richieste.

Lo svantaggio delle JSP è peró quello di tutti gli script CGI: la difficoltá sta nella manutenzione del codice HTML delle applicazioni complesse. Fortunatamente esiste una soluzione a questo problema e si chiama: *Java Bean*. I Java Beans sono dei componenti software che possono venire inclusi in una pagina JSP, permettendo quindi un ottimo incapsulamento del codice, pertanto riutilizzabile. Al programmatore sará pressochè invisibile la sezione di codice puro, che viene quindi sostituito da richiami ai metodi delle classi che costituiscono i Beans inclusi. Il risultato della compilazione dei file costituenti i Beans sono i file con estensione .class; questi file devono essere inclusi all'interno della pagina JSP che ne fa

utilizzo.

2.5 AJAX

Nella creazione dell'applicazione web RSSIWatcher si è fatto uso anche di questa particolare tecnica, la quale consente in poche parole di effettuare richieste di tipo HTTP ad un server web in modo indipendente dal browser. L'acronimo AJAX significa Asynchronous JavaScript and XML. L'entitá che sta alla base di questo concetto si chiama *XMLHttpRequest*: con questo, è possibile effettuare le richieste HTTP, avendo a disposizione anche la possibilitá di inserire i parametri necessari per la specifica richiesta. È inoltre possibile scegliere tra le due tipologie di richieste HTTP: GET o POST. Il concetto che sta alla base del funzionamento dell'AJAX è che la richiesta è asincrona: questo vuol dire che si deve attendere la terminazione di una richiesta prima di effettuare ulteriori operazioni che dipendono dai suoi risultati. Questo modo di pensare va contro il flusso dati tipico di una pagina web: con AJAX, mentre l'utente è all'interno della stessa pagina, possono venire lanciate numerose richieste, anche indipendenti l'una dall'altra, verso il lato server. Per ovviare al problema dell'attesa che intercorre tra l'invio di una richiesta e la risposta da parte del server, si puó far uso di una particolare funzione di callback, richiamata automaticamente dal sistema nel momento in cui lo stato di questa cambia. La richiesta infatti, durante il suo ciclo di vita, attraversa diverse fasi identificate da un numero intero; nel caso di RSSIWatcher è importante che la richiesta passi alla fase numero 4, che ne identifica la terminazione. Dopo il passaggio a questo stato, è possibile svolgere tutte le operazioni che erano in attesa della fine di ogni richiesta.

Capitolo 3

Architettura

In questo capitolo verrá descritta l'architettura dell'intero sistema utilizzato in questo lavoro di tesi; in particolar modo si parlerá dell'applicazione Android CSAMS e dell'applicazione web RSSIWatcher, andando a porre maggiore attenzione sulla struttura di quest'ultima. Verranno quindi descritti i diversi casi d'uso e le interazioni tra le varie componenti, considerando che parte delle scelte progettuali sono state obbligate dalla presenza di software esterni, come i moduli SPS e SOS; con le quali l'applicazione web e l'applicazione mobile CSAMS sono costrette ad interfacciarsi e ad interagire.

3.1 RSSIWatcher

L'applicazione cliente del sistema costruita ex-novo per espletare il servizio di monitoraggio del parametro RSSI è RSSIWatcher. Costruita con la tecnica delle JSP e con l'ausilio dei Java Beans, RSSIWatcher si interfaccia con i moduli SPS e SOS. Tali interazioni sono necessarie ai fini del funzionamento dell'applicazione, in quanto si devono fare delle interrogazioni per richiedere nuovi task nei confronti dei sensori registrati al sistema ed interrogazioni per recuperare le osservazioni da essi prodotte, in modo da presentare dinamicamente ad ogni avvio una panoramica del sistema con i valori piú aggiornati. RSSIWatcher pertanto, prima di cedere il controllo all'utente che la sta utilizzando, effettua una specie di "fase di caricamento", la quale puó essere pensata come una fase di "boot".

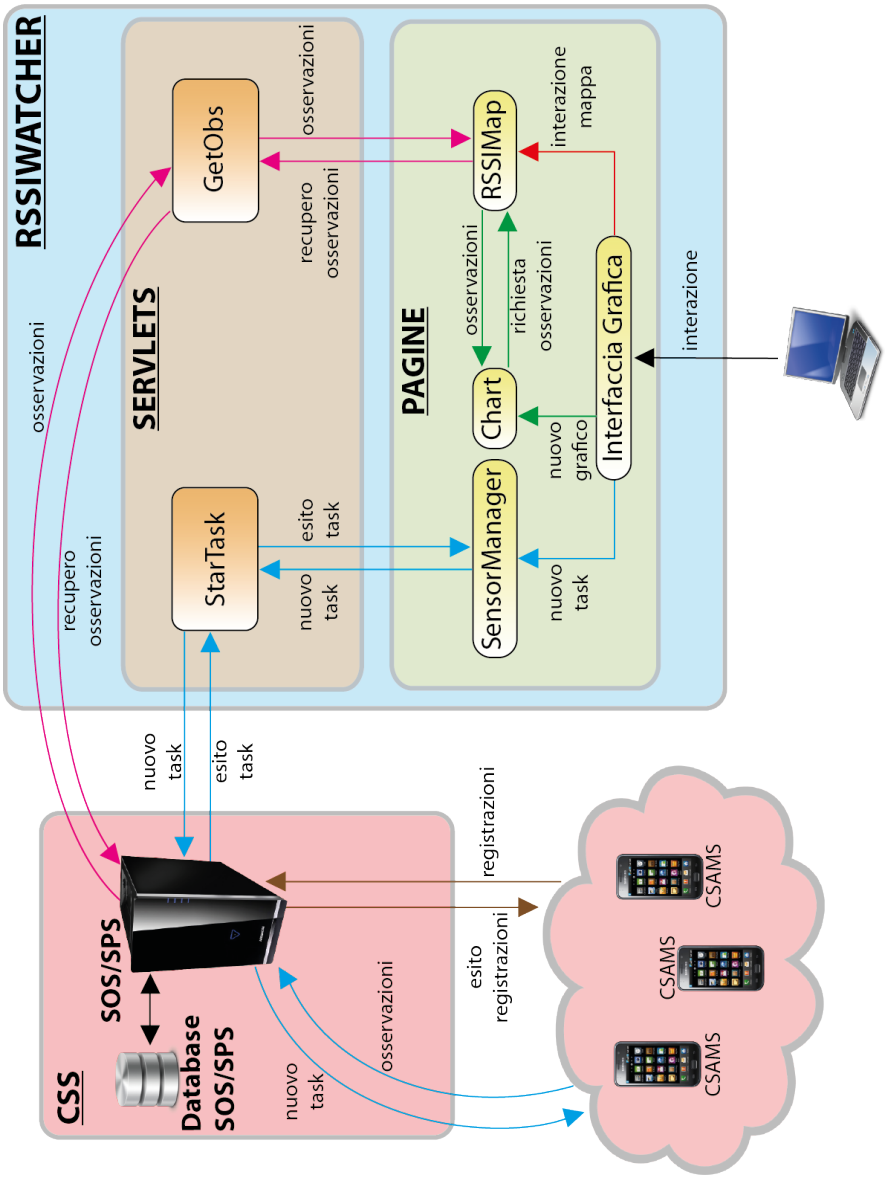

Figura 3.1: Architettura di RSSIWatcher

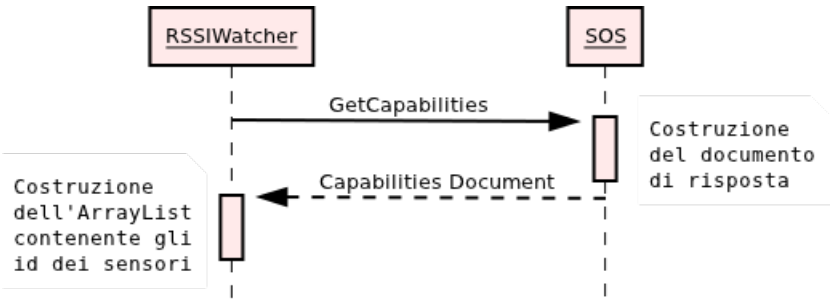

Figura 3.2: Sequenza operazioni per il recupero degli id

3.1.1 Fase di caricamento dei sensori registrati

Quando l'applicazione RSSIWatcher viene avviata dal web server Tomcat, prima di restituire dei risultati all'utente, ha bisogno di procurarsi tutte le informazioni del caso. Per prima cosa, si effettua una connessione verso il modulo SOS attraverso una richiesta di getCapabilities: il risultato di questa richiesta è un documento di risposta dal modulo SOS stesso, in formato XML. Il secondo passo è quindi quello di andare ad esaminare il documento XML ricevuto: questo conterrá informazioni riguardanti lo stato dell'istanza SOS interrogata. Tra le varie informazioni, RSSIWatcher filtra quelle che interessano allo scopo per il quale è stata costruita: nel documento vengono ricercati gli id dei sensori che si sono registrati per offrire task riguardanti il monitoraggio del parametro RSSI. Tali sensori sono facilmente riconoscibili nell'elenco di quelli registrati (si ricorda che ad una istanza di SOS si possono registrare diverse tipologie di sensori, anche quelli che non interessano a RSSIWatcher); infatti, in fase di registrazione, questi hanno spedito verso il server anche l'indice del fenomeno misurato. Questo indice identifica uno specifico sensore tra quelli gestibili dall'applicazione mobile CSAMS e in generale dall'intero sistema; nel caso del parametro RSSI, l'indice in questione è il numero 12. Gli identificativi presenti nell'istanza di SOS e SPS saranno quindi del tipo *12:numero-telefonico*. RSSIWatcher non deve far altro che andare a ricercare nel documento XML di risposta eventuali id che iniziano per "12:". Una volta trovati, questi vengono utilizzati per costruire una tabella che l'utente di RSSIWatcher puó consultare per scegliere eventualmente il prossimo sensore che deve effettuare un nuovo task. L'utente, grazie a questa tabella, puó inoltre rendersi conto del grado di partecipazione che si ha all'interno del sistema: avere tanti sensori registrati significa avere una buona accuratezza del parametro misurato, oltre che a una ottima copertura territoriale. Questa fase preliminare di

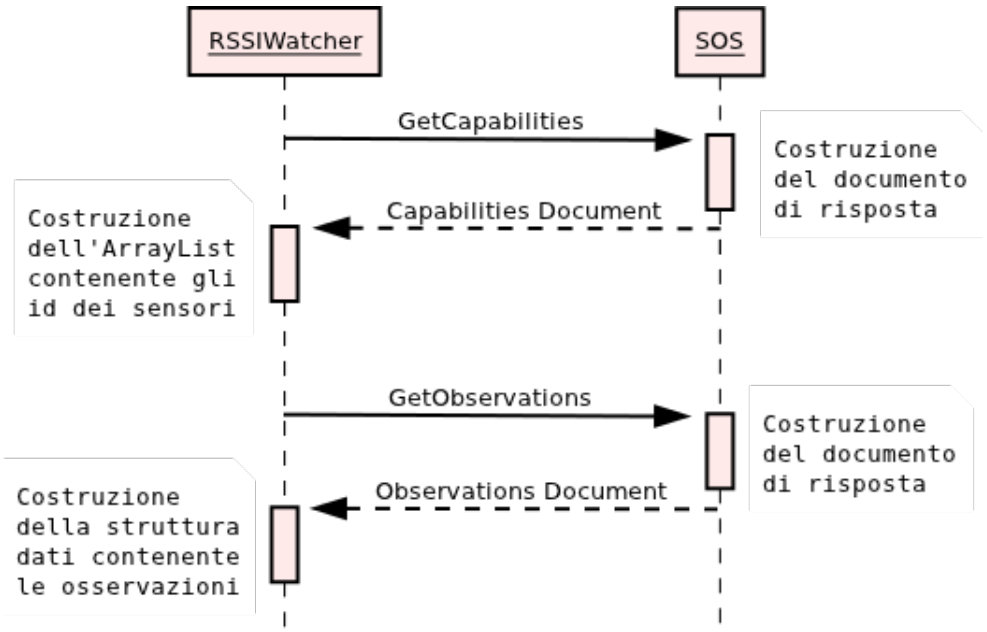

Figura 3.3: Sequenza operazioni per il recupero delle osservazioni

caricamento consente di avere ulteriori vantaggi nel momento in cui si ha la necessitá di inoltrare richieste successive: come si vedrá in seguito, RSSIWatcher ha bisogno anche di effettuare una richiesta di GetObservation verso il modulo SOS. Per effettuare questa richiesta sono peró necessari gli identificativi dei sensori. L'Array, contenente l'elenco di tali identificativi, viene quindi incluso nel file XML di richiesta di GetObservation in modo da ricevere come risultato le osservazioni riguardanti il fenomeno di interesse. In Figura 3.2 viene mostrata in un diagramma la sequenza di operazioni effettuate in questa fase.

3.1.2 Fase di caricamento delle osservazioni

Come la fase di caricamento dell'elenco dei sensori, RSSIWatcher ha bisogno di procurarsi a priori anche le osservazioni memorizzate nel database gestito dal modulo SOS e riguardanti la qualitá del segnale nella rete di telefonia mobile. Anche in questo caso, tale richiesta deve essere effettuata prima di restituire il controllo dell'interfaccia grafica all'utente che sta utilizzando l'applicazione e quindi costituisce una ulteriore parte della fase di "boot". La fase di caricamento delle osservazioni è fondamentale ed ha richiesto quindi particolare attenzione in fase di progettazione; per spiegare il motivo di questa esigenza, occorre descrivere il funziona-

mento del sensore GPS di un generico smartphone. Una volta attivato il rilevatore GPS, questo cerca di collegarsi con uno o piú satelliti in modo da creare con loro un collegamento stabile e ricevere quindi aggiornamenti corretti sul cambiamento della posizione. All'avvio di questo sensore è necessario attendere che l'allineamento con almeno un satellite sia completato e che quindi siano state ricevute almeno una coppia di coordinate latitudine e longitudine valida, prima di iniziare a spedire osservazioni verso il database di SOS. Inoltre, una volta collegato ad almeno un satellite e ricevuta quindi un informazione valida sulla locazione attuale, puó accadere che l'allineamento precedentemente stabilito venga temporaneamente a mancare. Questa situazione di instabilitá continua fino al momento in cui un nuovo allineamento non viene stabilito; questo lasso di tempo ha una durata non deterministica e dipende dalla capacitá del sensore di stabilire velocemente un nuovo collegamento. Prima di procedere oltre, occorre dire che la fase di caricamento delle osservazioni è stata pensata in modo che, una volta ricevuto il file XML di risposta dal modulo SOS, queste siano memorizzate localmente in un opportuna struttura dati all'interno della pagina. In questo modo, per qualsiasi necessitá, non occorre effettuare piú di una richiesta per manipolare i dati ricevuti, ma basta semplicemente navigare in questa struttura dati per reperire qualsiasi parametro delle osservazioni. Solo nel caso in cui si voglia cercare di scaricare una versione aggiornata delle osservazioni presenti nel database, questa struttura viene nuovamente riempita con i nuovi dati e quindi viene inviata una nuova richiesta verso il modulo SOS. In Figura 3.3 viene mostrata in un diagramma la sequenza di operazioni effettuate in questa fase.

3.1.3 Problemi legati al caricamento

Fatte queste premesse, vengono spiegate in seguito le motivazioni che hanno spinto a porre particolare attenzione nella fase di caricamento delle osservazioni. Il sensore GPS, il quale gioca un ruolo fondamentale nell'espletamento del servizio, deve fornire una coppia valida del tipo latitudine-longitudine, da allegare alle componenti delle singole osservazioni. Il suo utilizzo porta peró a varie problematiche che sono state previste e risolte giá nella fase di progettazione. Di seguito vengono descritti piú in dettaglio i punti cruciali ai quali ci si è trovati di fronte:

Coordinate non corrette Puó accadere che le coordinate geografiche, da inserire in una determinata osservazione, si trovino in uno stato inconsistente per svariati motivi e si deve quindi fare in modo di

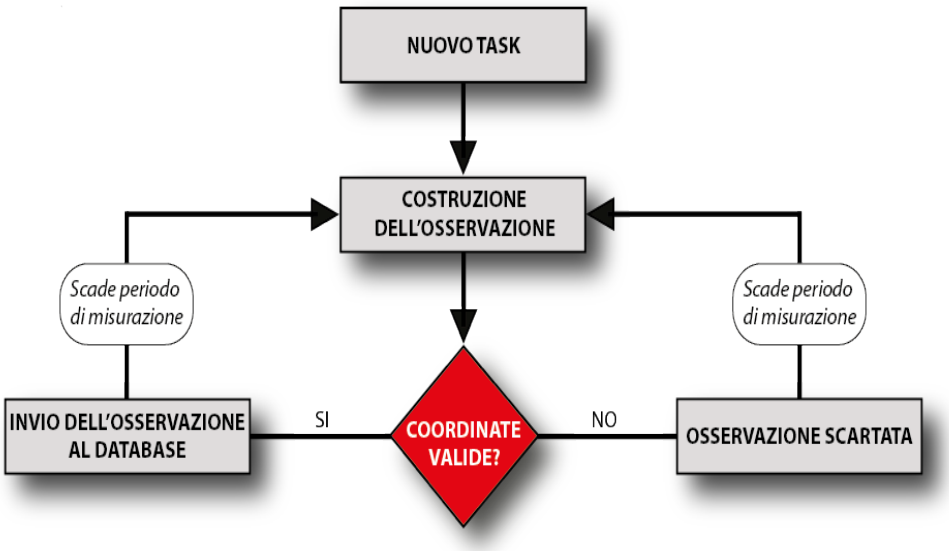

Figura 3.4: Diagramma di sequenza per la decisione sulle osservazioni

accorgersi del problema a monte. Vengono in seguito elencati nel dettaglio i casi in cui si ricade in questo tipo di inconsistenza sulle informazioni di localizzazione:

- Trovandosi in una posizione "scomoda" (come ad esempio all'interno di un edificio o comunque non perfettamente a "cielo aperto"), uno smartphone non riesce a soddisfare il task in arrivo con delle coordinate inizializzate in maniera oppotruna. In questo caso, essendo state inizializzate con valori uguali a zero, le coordinate restituite non sarebbero assolutamente valide.
- In questo caso l'applicazione ha attualmente un task attivo e sta cercando di recuperare le nuove coordinate per spedire l'osservazione successiva; ma, per una temporanea perdita di collegamento con il satellite, in memoria restano momentaneamente le vecchie coordinate. Tali coordinate, se fossero spedite cosí per come sono, potrebbero sembrare a prima vista corrette anche se in realtá non lo sono. Come giá detto, il lasso di tempo che costituisce la perdita del segnale, dipende dal dispositivo utilizzato; ma potrebbe essere abbastanza grande da includere lo scadere di uno o piú periodi di campiona-

mento, quindi per le nuove osservazioni verrebbero riutilizzate coordinate geografiche sbagliate in quanto non aggiornate.

- Un altro motivo per il quale le coordinate geografiche possono non essere molto accurate, è dovuto al fatto che un task puó essere avviato con un periodo di misurazione troppo corto; quindi il sensore GPS non ha il tempo materiale per rilevare la nuova posizione, anche se non ci sono stati problemi di perdita del segnale con il satellite.

Coordinate identiche ma corrette Come accennato precedentemente, trovandosi di fronte a osservazioni aventi le stesse identiche coordinate e dando per scontato che lo smartphone si trovi in mobilitá, si potrebbe pensare ad un comportamento errato da parte del sistema. Infatti, questo potrebbe essere causato da una momentanea perdita di allineamento con il satellite al quale si era collegati fino a poco tempo prima e verrebbe da pensare che si siano riutilizzate delle coordinate ormai obsolete. Bisogna peró stare attenti a una cosa: non per forza un utente di CSAMS si deve trovare in una situazione di perenne mobilitá, quindi non si puó pensare che un utente non possa mai trovarsi nella solita posizione per tutta la durata di un task. È pertanto normalissimo trovarsi di fronte a osservazioni diverse che abbiano le stesse coordinate (basta immaginare un utente di CSAMS che si trova seduto in un parco durante il task).

Dopo aver elencato i problemi precedenti, occorre spiegare come l'applicazione web RSSIWatcher riesca a risolvere parte di questi, in modo da non perdere nessuna osservazione fornita e quindi sfruttare a pieno tutte le informazioni presenti nel database SOS. In fase di caricamento, dopo aver prelevato tutti i dati dal database, RSSIWatcher controlla se ci sono osservazioni diverse con esattamente le solite coordinate geografiche. Se questo accade, si preoccupa di fornire come risultato una media aritmetica del parametro RSSI. Se questo tipo di accorgimento non fosse stato preso, sulla mappa, alle solite coordinate, sarebbero stati caricati piú marker: in questo modo peró si è notato che sarebbe stato visibile solamente l'ultimo inserito, mentre gli altri sarebbero rimasti nascosti. Con questa soluzione, viene garantito il fatto che tutte le osservazioni prodotte dai sensori contribuiscano in tutto e per tutto a fornire risultati per il servizio richiesto dall'utente finale. Per quanto riguarda il problema del periodo di misurazione troppo breve per permettere un cambio di coordinate, viene risolto in maniera molto semplice: l'utilizzatore dell'ap-

plicazione web deve essere consapevole delle spiacevoli conseguenze che si verrebbero a riscontrare nel caso in cui il periodo di misurazione immesso sia molto breve e sta quindi alla sua responsabilitá inserire un valore appropriato. È vero che un periodo di misurazione molto breve fornirebbe informazioni piú "fitte"; ma è anche vero che, oltre a causare i problemi giá esposti, un sensore, in un periodo molto breve, non puó essere in grado di percorrere un tratto di strada abbastanza lungo e due osservazioni molto vicine, geograficamente parlando, non avrebbero molto senso per il monitoraggio del parametro RSSI. Per ovviare al problema della perdita dell'allineamento con il satellite, entra in gioco la responsabilitá del possessore dello smartphone. Quest'ultimo, infatti, deve essere consapevole del fatto che rendersi disponibile per la richiesta di nuovi task da parte del suo sensore, in luoghi in cui è ovvio che un segnale GPS non sia facilmente instaurabile, non porterebbe ad alcun giovamento nei confronti del servizio offerto ai clienti dell'applicazione web. Resta quindi un ultimo problema da risolvere: cercare di non spedire osservazioni contenenti valori pari a zero nel campo latitudine e longitudine verso il database SOS. Quando questo accadesse, vorrebbe dire che CSAMS ha ricevuto una nuova richiesta di task e le coordinate geografiche devono essere ancora correttamente inizializzate. Sta quindi all'applicazione mobile rendersi conto di tutto questo fatto ed evitare di inoltrare tali osservazioni verso il modulo SOSPlugin. In Figura 3.4 viene mostrato in un diagramma di sequenza riguardante le operazioni effettuate in questa fase.

3.1.4 Assegnazione di marker e descrizione

I compiti che deve svolgere il modulo che si occupa del caricamento delle osservazioni non sono ancora finite: infatti, durante il caricamento della singola osservazione, oltre a prelevare il parametro RSSI, le coordinate geografiche, l'ID della cella e il parametro che indica l'accuratezza della misurazione fatta sulle coordinate, RSSIWatcher classifica ognuna di queste singole osservazioni assegnandogli un marker. Quest'ultimi sono stati costruiti appositamente per questa applicazione e sono stati colorati in maniera diversa a seconda del livello del segnale dell'osservazione associata. Inoltre, durante questa fase, viene assegnata ad ogni osservazione una descrizione. Tali descrizioni si dividono in due tipologie: quelle da assegnare alle osservazioni per le quali si è rivelato necessario effettuare una media sul parametro RSSI e quelle che semplicemente si compongono del contributo di una singola osservazione. Per la prima tipologia, la data che viene mostrata all'utente, è quella relativa alla primo rileva-

mento; si suppone infatti che, laddove il marker sia stato messo per piú osservazioni coincidenti, queste facciano parte di un solito task. Si da per scontato che sia pressochè impossibile che in un preciso punto vi si trovino due osservazioni effettuate da terminali diversi, in quanto le cifre dopo la virgola che compongono i parametri di latitudine e longitudine sono in gran numero. Per quanto riguarda i marker, si è deciso di crearne due diverse tipologie:

- **Prima tipologia:** questi marker sono adatti alla visualizzazione della mappa con un basso livello di zoom, essendo composti semplicemente da un pixel colorato in base al valore del parametro RSSI. Inoltre, per intravedere sotto di essi l' "ombra" della mappa, sono stati resi opachi. Si è deciso di dare al possibilitá all'utente, che sta utilizzando RSSIWatcher, di passare a questa tipologia di marker in quanto, considerando che la quantitá di osservazioni dovrebbe essere abbastanza alta, la mappa sembrerebbe "colorata" con delle sfumature (stile monti e pianure) in base all'intensitá del parametro misurato. In questo modo, l'utente, non interessato ai dettagli delle singole osservazioni prodotte, potrebbe avere una visione globale dell'andamento del segnale in un ampia area; proprio per questo motivo associata al marker non viene inserita nessuna descrizione.

- **Seconda tipologia:** si è scelto di aggiungere una seconda tipologia di marker per soddisfare anche gli utenti che abbiano una necessitá diversa rispetto a quella discussa nel punto precedente: un eventuale utilizzatore dell'applicazione RSSIWatcher potrebbe infatti avere interesse anche a esplorare una precisa area geografica e analizzare i valori che compongono le singole osservazioni. Tramite la pressione di un apposito bottone all'interno della mappa, è possibile quindi passare a quest'altra tipologia di marker, i quali sono stati disegnati in modo da rappresentare una specie di antenna con un pallino colorato in base all'intensitá del parametro. Insieme a questo tipo di marker viene anche associata una delle due descrizioni di cui si è discusso in precedenza; un utente quindi puó passare il puntatore del mouse sopra il marker per far comparire la descrizione, in modo da avere accesso a tutte le informazioni del caso. Quando viene passato il cursore del mouse sopra uno di essi, apparirá nella mappa una circonferenza avente per centro le coordinate del marker e per raggio il valore dell'accuratezza misurata sulle coordinate geografiche; in questo modo, l'utente puó render-

si conto in maniera visiva del grado di imprecisione sulla singola osservazione.

3.1.5 Tracciamento dei grafici

Adesso verrá introdotta una nuova funzionalitá offerta dall'applicazione RSSIWatcher. Nel caso in cui un utente fosse interessato a verificare se in un determinato lasso di tempo, piú o meno lungo, il parametro RSSI abbia avuto un comportamento anomalo all'interno di una certa area geografica, con le funzioni esposte fino ad ora si sarebbe trovato in difficoltá. Tale parametro, all'interno di una certa area geografica, potrebbe assumere valori particolarmente bassi e creare disagio nelle comunicazioni telefoniche, in seguito ad esempio ad un particolare fenomeno ambientale. Osservando semplicemente la cartina geografica, con la prima tipologia di marker, l'utente avrebbe una visione globale di quelle che sono state tutte le osservazioni prodotte dai vari sensori; non potrebbe avere, peró, la possibilitá di ricavare informazioni temporali in quanto, come giá detto precedentemente, non viene introdotta nemmeno una breve descrizione all'interno dei marker. Se si decidesse di passare alla seconda tipologia di marker le cose migliorerebbero, ma il problema non verrebbe ancora risolto: sebbene questa categoria di marker abbia al suo interno una descrizione dell'osservazione prodotta, sarebbe molto scomodo andare a ricavare la data esatta di produzione per ognuna di esse ed avere un idea di quale è stato l'andamento temporale del parametro. Per evitare questo tipo di fatica, all'utente che sta utilizzando l'applicazione viene offerta una comodissima funzionalitá: nell'interfaccia grafica viene proposto, tra l'altro, anche un semplice form da compilare. Tale form ha bisogno di ricevere una coppia di coordinate geografiche del tipo latitudine e longitudine e un raggio espresso in metri. Con questi dati, viene delimitata un area circolare che ha per centro il punto identificato dalle coordinate inserite e come raggio il numero espresso in metri. Per togliere all'utente anche la fatica di ricavarsi manualmente le coordinate di un preciso punto sulla mappa, queste vengono automaticamente inserite all'interno del form, attraverso un semplice click del mouse. Quando il form è stato correttamente compilato, RSSIWatcher ha tutti i dati necessari per il tracciamento di un grafico che tenga conto dell'andamento temporale del parametro osservato. Oltre a prelevare le coordinate geografiche, quando si effettua un click, all'interno della mappa viene disegnata un'area circolare che rappresenta l'area sulla quale verrá costruito il grafico. Sull'asse delle ascisse si trova l'informazione temporale, mentre sull'asse delle ordi-

nate si trova il valore in corrispondenza di una certa data del parametro RSSI, espresso in ASU (numero intero proporzionale alla potenza del segnale telefonico). L'utente puó inoltre andare a navigare all'interno del grafico per avere informazioni dettagliate sui singoli elementi che lo compongono. Occorre far notare una cosa importante: per effettuare questa operazione, non è necessario inoltrare nessuna richiesta di GetObservation verso il lato server. Quello che si sta cercando di dire è che le osservazioni che andranno a far parte del grafico costruito, si trovano giá memorizzate localmente all'interno della struttura dati, riempita durante la fase di "boot". Quello che RSSIWatcher fa al momento in cui un utente decide di tracciare un grafico, è quindi andare a scorrere la struttura dati costruita per verificare, direttamente da questa, quali inserire all'interno del grafico. Infine, occorre fare un ultima osservazione sulla creazione dei grafici temporali: una singola osservazione contiene, come giá detto, anche una coppia di coordinate che identificano il punto esatto in cui essa è stata effettuata. Questa rilevazione di coordinate geografiche, peró, è soggetta a una incertezza: Android fornisce un metodo che restituisce un valore espresso in metri indicante il raggio di incertezza. Quando viene definita dall'utente la zona circolare, sulla quale intende costruire il grafico temporale, alcune misurazioni potrebbero rimanere escluse ma il loro raggio di incertezza potrebbe farle comunque ricadere all'interno dell'area interessata dal grafico. Questa situazione è stata prevista ed è bastato includere nel calcolo della distanza anche il valore del raggio di incertezza sulle coordinate.

3.1.6 Iniziare un nuovo task

Un'altra funzionalitá molto importante offerta dalla web application è quella che permette all'utente di poter iniziare ex-novo un task; ovvero puó essere in grado di raggiungere un particolare dispositivo mobile, correttamente registrato al servizio, in modo da far iniziare una nuova campagna di task. Come accennato in precedenza, durante la fase di caricamento dei sensori registrati, viene costruito un Array contenente tutti i loro identificativi. Grazie a questo Array, sull'interfaccia grafica, viene disegnata una tabella contenente l'elenco. L'utente che sta utilizzando RSSIWatcher ha quindi a disposizione tutti i dati per poter iniziare un nuovo task: puó eventualmente scegliere, tra gli ID in tabella, quello da coinvolgere nel task e puó cominciare a riempire il form dedicato. Per completare tale form è necessario inserire anche il periodo di campionamento (stando attenti al fatto che un periodo di campionamento troppo

corto non permetterebbe un aggiornamento delle coordinate geografiche da parte del dispositivo mobile che effettua le misurazioni) ed infine la durata complessiva del task. Premendo il bottone di submit, la richiesta viene inoltrata al modulo SPS, dopo che RSSIWatcher ha terminato di costruire correttamente il file XML da inviare. SPS provvederá, grazie al suo plugin, ad inoltrare la richiesta al terminale mobile che si trova in attesa.

Va ricordato che, se un cliente di RSSIWatcher sta esplorando la mappa, con il secondo tipo di marker, ha la possibilitá di andare ad osservare la descrizione di una osservazione, per capire quale sensore l'ha prodotta. All'interno di tale descrizione, tra le varie informazioni, vi si trova anche una indicazione sul numero di osservazioni prodotte da tale sensore. Se l'utente ritiene che questo numero sia particolarmente basso, puó andare a formulare una richiesta di task da parte di questo sensore, andando quindi ad aumentare il grado di partecipazione di quest'ultimo all'interno del sistema.

3.1.7 L'utilitá della tabella dei sensori

A questo punto, dopo aver cominciato a capire quali sono le operazioni offerte da RSSIWatcher, verrebbe da pensare che sia presente un elemento ridondante all'interno dell'archietettura di questa applicazione web: la tabella dei sensori registrati, costruita durante una delle fasi di caricamento. Se ci si mette nei panni di un possibile cliente dell'applicazione, ci si potrebbe porre la seguente domanda: se nella mappa vi sono marker contenenti una descrizione completa, nella quale è visualizzato il numero di osservazioni effettuate da un particolare sensore, a cosa potrebbe essere utile la tabella dei sensori registrati al servizio? Per rispondere a questa domanda basta riflettere un attimo sul ruolo che gioca RSSIWatcher all'interno dell'intero sistema: considerando quali sono le componenti in gioco, si comprende che l'unica di esse capace di iniziare un nuovo task è proprio RSSIWatcher. Essendo l'unica capace di creare nuovi task, questa componente è anche l'unica che puó mostrare al cliente del sistema un elenco aggiornato dei sensori che si stanno registrando al servizio. È di fondamentale importanza ricordarsi che un sensore che si è appena registrato al servizio, sará visibile solamente all'interno della tabella. Sulla mappa, infatti, non si avrá visibilitá di questo sensore in quanto nessuno ancora puó avergli ordinato di effettuare un nuovo task. Pertanto, grazie alla tabella, che puó essere aggiornata in un qualsiasi momento con la pressione di un apposito bottone, un utente è in grado

di far entrare il nuovo sensore nell'insieme di quelli che hanno prodotto almeno una osservazione. La tabella dei sensori registrati al servizio è pertanto di fondamentale importanza ai fini del funzionamento dello stesso; senza di essa nessun nuovo sensore potrebbe produrre la sua prima osservazione (e ovviamente quelle successive). Le colonne che compongono tale tabella sono due: una dove sono contenuti gli identificativi di tutti i sensori e una dove, per ogni sensore, viene inserito il numero di osservazioni da lui effettuate fino a quel momento.

3.2 CSAMS

In questa sezione si cerca di descrivere il funzionamento dell'applicazione CSAMS, ma rispetto a come è giá stato fatto precedentemente, questa volta verrá posta maggiore attenzione ai compiti che dovranno svolgere i moduli coinvolti in una richiesta di osservazione sul parametro RSSI. Si tratta di moduli che hanno subito modifiche, o moduli che sono stati aggiunti per raggiungere lo scopo prefissato e quindi fanno parte di quelle componenti che si sono modificate o create durante il lavoro di tesi. L'applicazione CSAMS, nella sua versione iniziale e quindi prima di cominciare a ritoccare qualsiasi sua parte, aveva implementate al suo interno solamente le entitá necessarie alla gestione di due tipologie di sensori, diverse da quelle che si sarebbero dovute progettare in questo contesto. Per questo motivo, dopo aver esaminato a fondo il funzionamento globale dell'applicazione CSAMS per capire in quale punto intervenire, si sono dovute apportare le opportune modifiche all'interno della sua architettura. Tali modifiche sono state confinate solamente alle componenti che si trovavano ai livelli piú bassi dello stack dell'applicazione, la quale è stata progettata e realizzata appositamente con una struttura "a strati", proprio per permettere ad eventuali sviluppatori di effettuare modifiche future all'interno di essa e in modo tale da non andare a ritoccare l'intera struttura.

3.2.1 Il nuovo sensore

Prima di mettere le mani all'interno dell'implementazione di CSAMS, è stato necessario farsi un idea su come progettare il nuovo sensore per decidere i parametri da allegarvi per rendere completa e significativa una osservazione sul fenomeno misurato. Questo particolare tipo di sensore puó essere visto come la composizione di diversi "sottosensori" i quali, tutti insieme, forniscono una osservazione utile per l'applicazione web

RSSIWatcher. Pensandoci bene, se si volesse usufruire dei dati forniti da questo sensore, non sarebbe sufficiente un semplice numero che rappresenta la potenza del segnale ricevuto; questo infatti, da solo, non potrebbe essere associato per esempio a nessuna posizione geografica, o a nessuna posizione temporale. È necessario pertanto fermarsi a pensare a quali dati potrebbero essere interessanti per una applicazione cliente del servizio, come RSSIWatcher. Dopo una attenta riflessione, si è pensato che, oltre al parametro fondamentale RSSI, sarebbe stato utile contornare quest'ultimo con ulteriori informazioni:

- **ID della cella:** lo smartphone, come tutti i comuni telefoni cellulari, si trova in ogni momento (a meno di una cattivissima ricezione) in collegamento con una particolare antenna radio base, la quale definisce nel suo intorno una "cella". Tutti i telefoni cellulari, presenti in questo intorno, si troveranno molto probabilmente sotto la gestione di tale antenna. Aggiungendo questa informazione a ogni osservazione prodotta dal sensore, è possibile andare a scovare situazioni in cui si riscontra un cattivo funzionamento all'interno di una particolare cella. Per esempio potrebbe accadere che, periodicamente o in un determinato momento, una antenna rimanga fuori servizio; le osservazioni, prodotte dai terminali caduti sotto la gestione di tale cella, produrranno quindi osservazioni dove il parametro RSSI risulta di basso livello. Se tali osservazioni contengno anche l'ID della cella in questione, salterebbe subito all'occhio il problema di un possibile guasto dell'antenna e si potrebbe andare ad ovviare al problema in maniera piú mirata.

- **Coordinate geografiche:** si è deciso di introdurre anche questa coppia di valori per permettere di localizzare ogni singola osservazione. Come giá discusso in precedenza, avendo a disposizione questa informazione aggiuntiva, è possibile associare il valore RSSI ad un preciso punto all'interno di una mappa territoriale. È un parametro di fondamentale importanza in quanto, una applicazione cliente, non potrebbe sfruttare in nessun modo una osservazione contenente esclusivamente il parametro RSSI. Chiunque fosse interessato ad utilizzare tale sensore dovrebbe implementare, nella propria applicazione, la gestione delle mappe geografiche in modo da rendere visibili i vari risultati, proprio come è stato fatto per l'applicazione RSSIWatcher.

- **Raggio di incertezza:** a ogni osservazione prodotta si è pensato che fosse interessante associare il livello di precisione con la quale

è stata effettuata la rilevazione delle coordinate geografiche. Per fare questo è bastato inserire, nell'elenco dei parametri di una osservazione, anche il raggio di accuratezza asociato a una coppia latitudine-longitudine. Questo è stato possibile grazie ad un particolare metodo messo a disposizione da Android. Un cliente del servizio potrebbe servirsi di questo dato per farsi un idea di quando possa essere affidabile una particolare misurazione, oppure quanto possa essere in grado, un particolare sensore, di fornire informazioni precise dal punto di vista della posizione. Oltre a questi aspetti, questo valore è stato sfruttato durante la creazione dei grafici temporali in RSSIWatcher: una osservazione che non rientra nella zona circolare definita dall'utente, potrebbe rientrarvi nel caso in cui venisse considerato anche il raggio di incertezza e non solamente il punto definito dalle coordinate.

- **Marca temporale:** va ricordato che oltre a tutte le precedenti informazioni, l'applicazione fornisce per tutti i tipi di sensori anche una marca temporale. Essa indica in quale preciso momento è stata effettuata una osservazione e grazie a questo dato i cattivi comportamenti, in una particolare cella, potrebbero essere descritti anche con il loro andamento temporale.

3.2.2 Il Wrapper

Si è giá analizzato in precedenza il funzionamento generale dell'applicazione CSAMS e, come giá detto, il componente dell'applicazione che si occupa della gestione di un particolare sensore, effettuando le chiamate alle librerie che lo gestiscono, è il Wrapper. Il sensore utilizzato in questo lavoro di tesi, descritto nella sezione precedente, era sconosciuto nella prima versione di CSAMS. Pertanto, il secondo passo nella progettazione del nuovo sensore è stato quello di costruire e gestire il nuovo wrapper. I compiti che deve svolgere principalmente il Wrapper sono due:

- Gestire la richiesta ricevuta di nuovo task da parte del server e tradurla in chiamate a librerie che controllano e gestiscono i sensori fisici presenti nel dispositivo. Nel caso del wrapper utilizzato per le richieste di monitoraggio del parametro RSSI, le chiamate alle librerie che devono essere effettuate sono: quella relativa al parametro RSSI stesso, quella che preleva l'identificativo della cella alla quale si trova attualmente collegato ed infine quella che preleva le coordinate geografiche aggiornate.

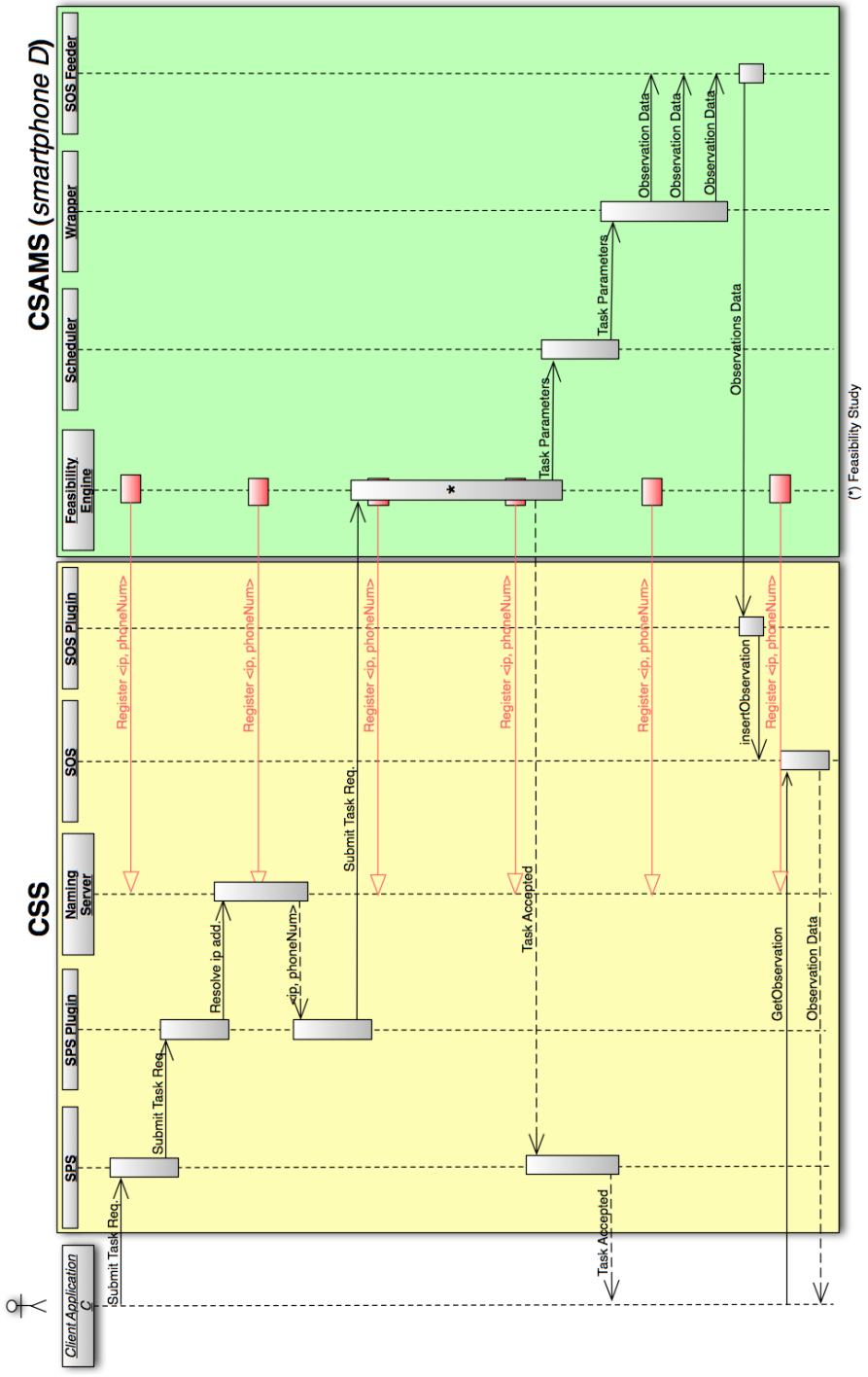

Figura 3.5: Diagramma casi d'uso del sistema globale

- Una volta avviato il task e dopo aver fornito i dati di una osservazione, il Wrapper deve porsi in attesa dello scadere del periodo di misurazione per effettuare l'iterazione successiva.

- Restituire i dati di ogni singola osservazione in un formato facilmente leggibile, in quanto si devono inserire questi parametri all'interno della richiesta che viene inoltrata al SOSPlugin. Si ricorda che, questa ultima componente è la responsabile della creazione dei documenti XML (contenenti le informazioni pervenute dagli smartphone) e della spedizione di quest'ultimi verso la componente SOS.

Durante la progettazione di questa componente ci si è trovati di fronte a una scelta progettuale. Il problema è nato dal fatto che a ogni osservazione deve essere associata a ogni costo una informazione di localizzazione; come giá detto, gli smartphone, nella gestione del proprio GPS, possono presentare un certo ritardo prima di agganciarsi correttamente a un satellite e quindi avere delle informazioni corrette su latitudine e longitudine attuali. Dovendo in questo caso far uso del sensore GPS, si è presentato il seguente quesito: all'arrivo di una richiesta di task, è preferibile porsi in attesa di un segnale GPS valido prima di inizare un task? oppure è meglio non inserire nel database osservazioni con latitudine e longitudine pari a zero, rispettando comunque i vincoli temporali imposti dalla richiesta? La risposta a tale domanda si basa sulle seguenti considerazioni:

- L'informazione sulla posizione è di fondamentale importanza affinché il parametro RSSI abbia un significato utilizzabile da terze applicazioni clienti e quindi deve essere per forza presente.

- Le richieste pendenti possono nuocere al funzionamento del sistema. Accettare una richiesta mantenendola in attesa aspettando un segnale GPS valido, potrebbe infatti impedire l'arrivo di altre richieste di diverso tipo che potrebbero essere immediatamente soddisfatte in quanto non necesariamente devono fare uso del GPS.

- Il segnale GPS, a meno che non ci si trovi all'interno di edifici o in particolari luoghi non a "cielo aperto", diviene pronto in breve tempo. Come detto in precedenza inoltre, non si puó essere interessati al comportamento di tale parametro con una alta frequenza di campionamento; ma soprattutto, generalmente, se si chiede di effettuare un nuovo task a uno smartphone, questo deve avere una

durata ragionevole. Un osservazione di 10 secondi, per esempio, non avrebbe alcun significato pratico. Sapendo quindi che le osservazioni di questo fenomeno non devono avere una frequenza di campionamento molto elevata e che la durata non puó essere breve, si puó arrivare alla conclusione che la perdita di alcune osservazioni all'interno di un task non compromette il task stesso.

- Sta alla consapevolezza del possessore dello smartphone, avviare l'applicazione CSAMS nel momento in cui il proprio segnale GPS possa diventare valido in breve tempo.

Detto questo, soprattutto in considerazione del secondo e del terzo punto, si puó rispondere alla domanda precedente dicendo che è preferibile lasciare che il task inizi. Il compito di decidere se inoltrare le singole osservazioni verso l'SOSPlugin viene assegnato ad un componente di CSAMS progettato appositamente per questo scopo: il *CSAMSRssiManager*. Quest'ultimo effettua degli inserimenti condizionati nel database (nel prossimo capitolo verranno forniti ulteriori dettagli in proposito). In Figura 3.5 viene rappresentato un diagramma che descrive come interagiscono tra loro tutte le componenti del sistema.

Capitolo 4

Implementazione

In questo capitolo, verranno descritti i dettagli implementativi dei moduli che compongono l'applicazione web RSSIWatcher e quelli che si sono dovuti modificare e creare ex-novo per rendere disponibile il nuovo sensore all'interno dell'applicazione Android CSAMS.

4.1 RSSIWatcher

Per quanto riguarda l'applicazione web RSSIWatcher, si ricorda che questa è stata implementata con la tecnica delle Java Server Pages e con l'ausilio dei Java Bean: componenti da utilizzare nel momento in cui si vuole rendere il codice piú leggibile e si vuole avere una separazione piú netta tra la parte di codice responsabile dell'intero funzionamento del programma e quella parte di codice che invece si preoccupa di presentare i risultati all'utente. All'interno dell'applicazione RSSIWatcher non si è fatto solamente uso di codice Java, si sono utilizzati infatti anche:

- **HTML:** si è fatto uso di questo linguaggio per creare le componenti che forniscono l'interazione tra l'utente e l'applicazione web (come i FORM) e per dividere la pagina iniziale in tre FRAMES, i quali contengono a loro volta altre pagine JSP o HTML che compongono l'applicazione stessa.

- **Javascript:** l'utilizzo del Javascript è stato forzato dal fatto che le API, rese disponibili da Google per la gestione delle mappe e dei grafici, erano scritte proprio con questo tipo di linguaggio. L'alternativa che si era presentata all'inizio era quella dell'utilizzo delle

Static API di Google. Questo tipo di API prevedeva peró che l'utente dovesse inserire, all'interno di una richiesta HTTP di tipo GET, tutti gli elementi che desiderava far apparire all'interno della mappa: ad esempio, se fosse stato interessato a inserire tutti i marker che costituiscono le osservazioni, avrebbe dovuto specificarli una alla volta all'interno di una sola richiesta di tipo GET. Questa tecnica portava con se un grosso handicap: la richiesta HTTP di tipo GET presenta un limite sul numero dei caratteri da poter inserire. Pertanto, nel contesto di questo lavoro di tesi, il problema si sarebbe sicuramente presentato, in considerazione del fatto che le osservazioni prodotte sono numerose. Inoltre, le API Javascript forniscono molte operazioni in piú rispetto alle Static API, le quali si suppone debbano essere utilizzate in ambiti in cui la gestione delle mappe non costituisce un punto cruciale all'interno del sistema.

- **XML:** questo tipo di documenti si sono utilizzati in quanto sono richiesti nel momento in cui si devono effettuare delle interazioni con i moduli residenti sul lato server (SPS e SOS).

4.1.1 I Java Beans

In questa sezione verranno descritti i Beans utilizzati all'interno delle pagine JSP, per rendere il codice delle stesse il piú pulito possibile e per separare in maniera netta la parte implementativa dalla parte di visualizzazione dei risultati. Si ricorda che i Java Beans non sono altro che delle classi scritte in linguaggio Java, le quali verranno istanziate all'interno di una pagina JSP in modo da rendere disponibili le loro funzioni all'interno di essa.

4.1.2 GeneralSettings

In questa classe si è deciso di inserire alcuni parametri statici che, all'interno dell'applicazione RSSIWatcher, venivano richiesti di frequente. All'interno di essa sono contenute solamente variabili di tipo stringa che rappresentano URL o percorsi all'interno delle directory della macchina sul quale è installato il sistema. Inserire direttamente tali parametri all'interno del codice delle pagine JSP che li utilizzano, si sarebbe rivelata una soluzione poco efficiente nel momento in cui si fosse resa necessaria una modifica su uno di essi: si sarebbero dovuti andare a cercare infatti tutti i punti in cui le variabili venivano utilizzate, in modo da modificarle con il valore aggiornato. Questa operazione, eseguita in questo modo,

diventa peró molto complicata e dispendiosa in termini di tempo. Sfuttando il Bean GeneralSettings, è sufficiente andare solamente a mettere mano all'interno della classe che contiene tutte queste variabili, in modo da cambiarle una volta per tutte, senza andare a mettere mano all'interno di tutte le pagine JSP che le utilizzano. Le variabili contenute in questa classe sono le seguenti:

- **SOSUrl:** un riferimento all'installazione del modulo SOS. In questo lavoro di tesi l'installazione di Tomcat contiene sia i moduli SOS e SPS che l'applicazione web RSSIWatcher; pertanto, il suo valore è *http://localhost:8080/52nSOSv3/sos*. Questa particolare stringa viene utilizzata all'interno dell'applicazione web ogni volta che si rende necessaria una interazione con il modulo SOS.

- **SPSUrl:** vale lo stesso discorso fatto al punto precedente, ma per quanto riguarda il modulo SPS. Il suo valore all'interno di questo lavoro di tesi è quindi *http://localhost:8080/MYSPS/SPS*.

- **getCapString:** la richiesta di GetCapabilities nei confronti del modulo SOS puó avvenire anche effettuando una richiesta HTTP di tipo GET (puó essere utilizzato anche il metodo POST, ma in questo modo viene risparmiata qualche riga di codice). Tale stringa viene utilizzata quindi interno di RSSIWatcher ogni volta che si rende necessaria una richiesta di GetCapabilities. Basta quindi concatenare questa stringa con quella contenuta all'interno di SOSURL per effettuare la richiesta nei confronti del modulo SOS. Il suo valore è quindi *?REQUEST = GetCapabilities&SERVICE = SOS&ACCEPTVERSIONS = 1.0.0*.

- **submitUrl:** questa stringa contiene un percorso all'interno delle directory della macchina che ospita l'intero sistema. All'interno di una particolare directory devono essere inseriti infatti dei documenti XML, da riempire in maniera opportuna e da spedire verso i moduli SOS e SPS durante le comunicazioni tra l'applicazione web e quest'ultimi. Tali documenti XML contengono dei particolari tag da sostituire con i parametri veri e propri delle richieste. All'interno di questa particolare stringa viene memorizzato l'URL da utilizzare per reperire il documento XML, necessario per effettuare una richiesta di SUBMIT verso il modulo SPS.

- **getObsUrl:** anche per questa stringa vale il discorso fatto al punto precedente, in questo caso peró la variabile contiene il percorso da

effettuare per reperire il documento XML da utilizzare nel momento in cui si rende necessaria una richiesta di GetObservations verso il modulo SOS.

4.1.3 ConnectionUtils

Questa classe costituisce il Bean che gioca il ruolo principale nel meccanismo dell'applicazione RSSIWatcher. È infatti questo componente che garantisce il corretto funzionamento dell'applicazione web, fornendo le funzioni necessarie per effettuare in maniera opportuna le connessioni verso i moduli SOS e SPS. Basta quindi istanziare un Bean di questo tipo all'interno delle pagine JSP per completare le interazioni tra l'applicazione web e il lato server. In seguito vengono elencati i metodi utilizzati all'interno di questa classe:

- *public ArrayList<String> getCapabilities(String peerName,String url):* questa operazione viene richiamata dalle pagine JSP nel momento in cui vogliono effettuare una richiesta di getCapabilities verso il modulo SOS. I parametri richiesti da tale funzione sono:
  - un nome che identifica il peer coinvolto (in questo caso SOS).
  - l'URL per raggiungere l'installazione del modulo SOS. Tale URl viene utilizzato per aprire la connessione HTTP.

  Dopo aver settato la richiesta HTTP di tipo GET in maniera opportuna, viene aperta una connessione verso l'URL specificato nella richiesta. Al completamento di quest'ultima, il metodo ricava il *ResponseCode* e controlla se questo contiene il numero 200: in tal caso significa che la richiesta è stata inoltrata correttamente e quello che resta da fare è quindi effettuare una chiamata alla funzione *manageGetCapResponse* passando come parametro la risposta ricevuta dal modulo SOS. Nel caso in cui il codice di risposta non sia il numero 200, viene invece lanciata una eccezione che serve a segnalare la presenza di un errore durante la fase di transazione.

- *private ArrayList<String> manageGetCapResponse(String response):* questo metodo viene richiamato subito dopo aver effettuato una richiesta di GetCapabilities verso il modulo SOS. L'unico parametro richiesto da questa funzione rappresenta il corpo della risposta ad una richiesta di GetCapabilities. Alla fine della richiesta, se il response code è il numero 200, viene restituito dal modulo SOS un file XML all'interno del corpo della risposta. Il compito

fondamentale di questa funzione è quindi quello di gestire il documento XML di risposta, in modo da isolare in un ArrayList l'elenco degli identificativi dei sensori che si sono registrati. Si ricorda che, all'interno del sistema CloudSensor, ogni tipologia di sensore è identificato da un numero intero: nel caso di questo lavoro di tesi, il sensore da utilizzare è identificato dal numero 12. All'interno di questo metodo si fa largo uso della funzione *split()*, offerta da Java per la manipolazione delle stringhe e si cerca di isolare all'interno del documento gli identificatori dei sensori che iniziano con la sottostringa *12-*. Quando ne viene trovato uno, questo viene inserito all'interno dell'ArrayList, il quale, come si nota dall'intestazione dalla funzione, viene restituito all'interno delle pagine JSP, le quali possono quindi scorrerlo per consultare l'elenco dei sensori.

- *public String startTask(String peerName,String docUrl,String sensorId,String duration,String period):* questo metodo viene invocato dalle pagine JSP che intendono far iniziare un nuovo task ad un particolare sensore. I parametri utilizzati in questa richiesta sono:
  - il nome del peer coinvolto nella richiesta (in questo caso il modulo SPS).
  - il percorso da utilizzare per recuperare il file XML da riempire con i parametri della richiesta. Tale documento deve essere inserito all'interno della richiesta HTTP di tipo POST da inviare al modulo SPS, il quale la inoltrerá di conseguenza verso il sensore coinvolto.
  - l'URL per raggiungere l'installazione del modulo SPS. Tale URl viene utilizzato per aprire la connessione HTTP.
  - l'ID del sensore coinvolto nel nuovo task.
  - la durata del nuovo task.
  - il periodo del nuovo task.

  La prima operazione che effettua tale metodo è la costruzione di un documento XML che contenenga tutti i parametri necessari per effettuare una nuova richiesta di task. In questo documento devono essere quindi inseriti tutti i valori che caratterizzano il task stesso (periodo di campionamento e durata) e l'ID del sensore registrato al sistema che si vuole coinvolgere. L'URL contenuto all'interno del parametro *docUrl* contiene il percorso da fare all'interno delle directory per reperire il documento XML, all'interno del quale si

trovano alcuni TAG. Al posto di tali TAG, la funzione startTask deve inserire i valori numerici necessari per completare il documento. Dopo questa operazione, si preoccupa di settare correttamente la connessione, in modo da inviare la richiesta HTTP di tipo POST contenente il documento appena costruito; dopo di che attende la risposta dal modulo SPS. Come nel caso precedente, se il respose code associato a tale richiesta è il numero 200 significa che tutto è andato a buon fine e che la richiesta è stata inoltrata con successo; a questo punto si puó effettuare una chiamata alla funzione *manageSubmitResponse*. In caso contrario, significa che qualcosa è andato storto durante la transazione e viene quindi lanciata una eccezione.

- *private manageSubmitRespose(String response):* questo metodo viene richiamato subito dopo aver completato correttamente una richiesta di Submit nei confronti del modulo SPS. L'unico parametro richiesto da questa funzione rappresenta il corpo della risposta ricevuto in seguito alla richiesta. Alla fine di un'operazione di Submit, nel caso in cui il response code sia 200, viene restituito un file XML contenente alcune informazioni riguardanti l'esito della sottomissione della richiesta. Questa funzione si occupa pertanto di navigare all'interno del corpo di tale risposta per andare a reperire ulteriori informazioni. Anche se il codice di risposta è stato il numero 200 questo non significa che il task è stato sottomesso correttamente o che sia giá iniziato, ma significa solo che la richiesta è stata ricevuta correttamente dal modulo SPS. Questo peró potrebbe non essere in grado di far iniziare il nuovo task per vari motivi, come per esempio quando si rende conto che l'ID del sensore inserito nel documento XML è errato. La prima operazione che questa funzione esegue è quella di andare a cercare all'interno del corpo della risposta la presenza di una sottostringa del tipo *ExceptionText* o *Status*. Nel primo caso, se la sottostringa è presente all'interno del documento, significa che il task non è stato iniziato correttamente e quindi la richiesta non è andata a buon fine; nel secondo caso, significa che il task è stato inoltrato correttamente e vengono forniti ulteriori dettagli sullo stato dello stesso. ManageSubmitRespose si occupa infine di restituire all'utente della web application una stringa da visualizzare su schermo per fornire informazioni sullo stato del task che ha appena inoltrato.

- *public String getObs(String peerName,ArrayList<String> sensors,*

String url, String docUrl): questa funzione viene invocata dalle pagine JSP che hanno l'intenzione di effettuare una richiesta di GetObservation verso il modulo SOS. I parametri richiesti da questo metodo sono i seguenti:

- il nome del peer coinvolto nella richiesta (in questo caso il modulo SOS).
- l'ArrayList contenente gli id dei sensori per i quali si desidera recuperare le osservazioni effettuate.
- il percorso da fare per recuperare il file XML da riempire in maniera opportuna con i parametri della richiesta. Tale documento deve essere inseririto nella richiesta HTTP di tipo POST da inviare al modulo SPS per iniziare il nuovo task.
- l'URL per raggiungere l'installazione del modulo SPS. Tale URl viene utilizzato per aprire la connessione HTTP.

Come giá detto piú volte, quando si desidera effettuare richieste verso i moduli SOS e SPS, si devono costruire degli appositi documenti XML contenenti le informazioni necessarie per effettuare la richiesta stessa. Anche in questo caso, la prima cosa da fare all'interno di questa funzione è quindi recuperare dall'ArrayList ogni singolo ID, in modo da inserirli opportunamente all'interno del documento XML da costruire. La sostituzione avviene utilizzando anche questa volta dei TAG da rimpiazzare con spezzoni di codice XML contenenti i valori numerici degli identificativi dei sensori. Dopo aver effettuato la sostituzione, il documento XML è pronto per essere spedito; viene quindi aperta la connessione e viene effettuata una richiesta HTTP di tipo POST e il documento XML viene ovviamente inserito nel corpo di quest'ultima. Come avviene per i precedenti casi, al termine della richiesta, si verifica il response code: se questo è identificato dal numero 200 significa che la richiesta è stata inoltrata correttamente e viene immediatamente effettuata una chiamata alla funzione *manageGetObsResponse*. In caso contrario significa che c'è stato un errore in fase di transazione e viene lanciata una eccezione.

- *private String manageGetObsResponse(String response):* questo è il metodo che viene invocato dopo la richiesta di GetObservation nei confronti del modulo SOS per gestire la risposta proveniente da quest'ultimo. Il corpo di tale risposta è costituito sempre da un documento XML, il quale contiene le osservazioni effettuate dai

sensori passati come parametro della richiesta. L'unico valore da passare a questo metodo è costituito dal corpo della risposta. Il compito fondamentale di questa funzione è quello di restituire in un formato "compatto" tutte le osservazioni, separate da opportuni caratteri separatori. Si è scelto di creare una stringa compatta con tutte le osservazioni, in quanto questa dovrá essere restituita all'interno del codice Javascript che ha invocato la richiesta; in questo modo sará possibile creare in maniera molto semplice e veloce la struttura dati da mantenere in locale, la quale puó essere consultata per recuperare in futuro i valori che costituiscono le varie osservazioni (in seguito verranno forniti dettagli a riguardo). La stringa "compatta" sará formattata in modo da contenere, per ogni sensore, un informazione riguardante il numero di osservazioni prodotte da esso, seguita dalle osservazioni stesse.

4.1.4 Le pagine HTML e JSP

Dopo aver descritto i Java Beans da inserire all'interno delle pagine JSP, si passa alla descrizione dei dettagli implementativi riguardanti le pagine che costituiscono l'applicazione web RSSIWatcher. Queste rappresentano l'interfaccia grafica attraverso la quale un eventuale utente interagisce con essa, effettuando richieste verso il lato server o semplicemente consultando i risultati ottenuti dalle richieste precedenti.

4.1.5 Index.html

Questa pagina contiene esclusivamente codice HTML e rappresenta la "cornice" dell'applicazione. Per ottenere una interfaccia grafica di facile intuizione e il piú possibile compatta, si è deciso di dividere la pagina principale in tre frames. Questi frames ospiteranno una ulteriore pagina (HTML o JSP), la quale fornirá all'utente la possibilitá di effettuare un determinato tipo di operazioni. Di seguito vengono descritti ognuno dei tre frames, introducendo le varie operazioni che mettono a disposizione dell'utente:

- Nella sezione in alto a sinistra viene riservato un frame dedicato alla gestione del "lato sensori". In questa sezione, un utente puó infatti reperire informazioni riguardo i sensori registrati al sistema, oppure puó interagire con essi inviando nuove richieste di task.
- Nella sezione in basso a sinistra si trova invece un frame dedicato alla gestione dei grafici. Come giá accennato in precedenza, l'ap-

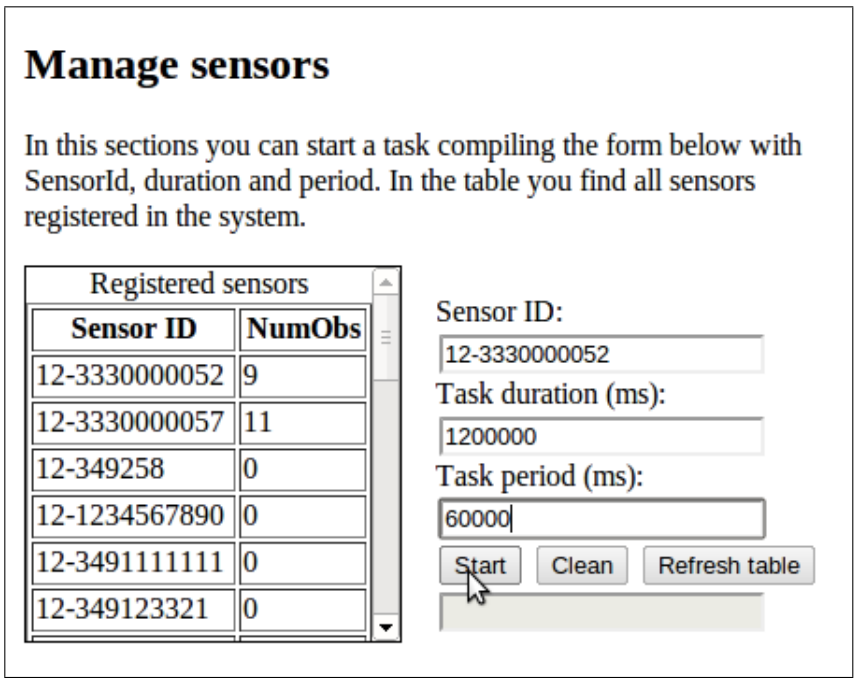

Figura 4.1: Screenshot della pagina dedicata ai sensori

plicazione RSSIWatcher fornisce all'utente la possibilitá di disegnare grafici che rappresentino l'andamento temporale del parametro RSSI all'interno di una determinata area geografica.

- La parte destra dello schermo è invece interamente dedicata alla gestione della mappa geografica. L'utente, in questo frame, puó consultare quindi i risultati ottenuti dopo una richiesta di GetObservation, andando ad osservare nel dettaglio le descrizioni che si trovano all'interno di ogni marker. Oppure si puó ottenere una visione globale dell'andamento del parametro RSSI, grazie all'utilizzo dell'altro particolare tipo di marker in precedenza descritto.

4.1.6 SensorManager.jsp

Questa pagina ha l'estensione di una Java Server Page in quanto, all'interno di essa, si trovano spezzoni di codice HTML alternati a spezzoni di codice Java per fornire una presentazione dei risultati in maniera dinamica. Questa pagina è quella che viene caricata all'interno del frame che si trova in alto a sinistra. Essa fornisce all'utente la possibilitá di gestire e consultare i sensori che si sono correttamente registrati al servizio;

al suo interno, si ha inoltre la presenza del linguaggio Javascript. Questo particolare linguaggio viene utilizzato ogni volta che si ha a che fare con AJAX (ampiamente discusso in precedenza), il quale a sua volta viene utilizzato quando si ha la necessitá di effettuare richieste di tipo HTTP.

All'interno di SensorManager.jsp viene utilizzata la tecnica dei Java Beans, in modo da nascondere il piú possibile la parte di codice che si occupa del lato applicativo. Verranno descritti adesso quali sono i compiti che deve svolgere questa pagina e quali sono le operazioni che deve effettuare prima di rilasciare il controllo dell'interfaccia grafica all'utente. In Figura 4.1 viene mostrato lo screenshot della pagina dedicata alla gestione dei sensori.

Creazione della lista dei sensori

La pagina SensorManager.jsp contiene al suo interno una tabella con gli identificativi dei sensori registrati al servizio e il numero delle osservazioni da loro effettuate. Questa particolare tabella viene costruita nel momento in cui la pagina viene caricata e quindi l'utente non ha il controllo sulla sua creazione. Per creare la tabella viene effettuata una chiamata alla funzione getCapabilities del Bean ConnectionUtils. Grazie a questa chiamata, che viene effettuata ogni volta che si ha il refresh della pagina, si è in possesso dell'Array contenente la lista degli identificativi dei sensori registrati. Il prossimo passo è quello di andare a scorrere tale Array e creare in maniera dinamica la tabella dei sensori, mischiando all'interno di un ciclo *for* sia il linguaggio Java che il codice HTML, quest'ultimo necessario per costruire la tabella stessa.

Possibilitá di nuovi task

Dopo aver terminato di costruire la tabella dei sensori registrati al servizio, il controllo dell'interfaccia viene restituito all'utente, il quale ha la possibilitá, all'interno di tale pagina, di riempire un semplice form inserendo i parametri necessari per iniziare un task ex-novo. In particolare i parametri richiesti sono:

- La durata complessiva del task che si intende iniziare.
- Il periodo di campionamento di tale task.
- L'identificativo del sensore che deve iniziare il task.

Una volta finito di compilare il form, l'utente puó premere il bottone di Submit per avviare la procedura che porta all'avvio di un nuovo task.

In particolare, si effettuerá una chiamata alla funzione Javascript *startTask()*, la quale si deve preoccupare di effettuare gli opportuni controlli sui singoli campi del form, verificandone sia la presenza che la correttezza dal punto di vista sintattico. Se questa fase di verifica va a buon fine, il passo successivo di tale funzione è quello di settare correttamente una connessione con la tecnica AJAX. In questo modo è possibile inoltrare una richiesta ad una servlet creata appositamente per svolgere tale compito. La servlet in questione si chiama *StartTask.jsp* e verrá discussa nel dettaglio in seguito. All'interno dell'URL della richiesta HTTP di tipo GET vengono inseriti i parametri del nuovo task; una richiesta verso questa particolare servlet dovrebbe avere questa forma: *http://localhost:8080 / RSSIWatcher / StartTask.jsp ? sensorId = id-sensore & period = periodo-task & duration = durata-task*. Facendo uso dell'AJAX, la richiesta viene completata in maniera asincrona, pertanto è stato necessario inserire una funzione di callback invocata automaticamente quando la richiesta passa alla fase 4 (numero che sta ad indicare il suo completamento). Grazie a questa funzione l'utente viene a conoscenza dell'esito della richiesta da lui inoltrata.

4.1.7 StartTask.jsp

All'interno di questo file si trova solamente codice Java; pertanto, questa puó essere definita come una servlet che non ha bisogno di presentare a video particolari risultati. Essa viene invocata attraverso una richiesta effettuata con la tecnica AJAX, da parte del SensorManager, il quale attende il suo completamento per mostrare l'esito all'utente che sta utilizzando RSSIWatcher. All'interno di questa JSP si fa utilizzo dei Java Beans; in particolare, è necessario creare un istanza del Bean ConnecionUtils per permettere di utilizzare la funzione che apre una connessione verso il modulo SPS: *startTask*. Il risultato di tale funzione è una stringa che descrive l'esito della richiesta. All'interno di questo Bean, per restituire un informazione sull'esito, si effettua una chiamata alla funzione *out.println()*. L'invocazione di questo metodo all'interno di una Servlet permette di restituire, a chi ha effettuato la richiesta, particolari risultati. La stringa ritornata, viene prelevata dalla funzione Javascript che l'ha invocata grazie a questa semplice riga di codice: *var resp = request.responseText;*.

4.1.8 Chart.html

Questa pagina è stata progettata per permettere all'utente che sta utilizzando l'applicazione RSSIWatcher di disegnare un grafico di tipo scatter, il quale mostra l'andamento temporale delle osservazioni sul parametro RSSI effettuate dai vari smartphone. La scelta di costruire una pagina di questo tipo è stata presa in considerazione del fatto che, osservando semplicemente la mappa geografica con tutte le osservazioni disegnate, non sarebbe possibile farsi un idea di come queste siano distribuite nel tempo; ma piuttosto si riesce solo ad osservare come si comporta l'andamento del segnale all'interno di un area. Occorre ricordare che, in questa pagina, per costruire un grafico, si deve preventivamente compilare un semplice FORM contenente alcune informazioni necessarie a restringere la rappresentazione del grafico stesso nell'area geografica di interesse. Si è scelto quindi di progettare il FORM in modo che contenesse come campi sia un punto espresso in coordinate geografiche con una coppia latitudine-longitudine che un raggio espresso in metri: questi tre parametri, insieme, definiscono quindi un area geografica di forma circolare. Il grafico mostrerá quindi solamente i contributi delle osservazioni che ricadono all'interno questa particolare area. In tale pagina si trova sia il codice HTML che il codice Javascript; la presenza di quest'ultimo è obbligata dal fatto che si è deciso, anche in questo caso, di utilizzare le API offerte da Google per gestire e manipolare diverse tipologie di grafici, in quanto risultano di semplice utilizzo. Tornando ai dettagli implementativi di questa pagina, il controllo dell'interfaccia grafica viene subito restituito all'utente e le funzioni presenti all'interno della pagina sono le seguenti:

- *drawChart():* una volta compilato il FORM relativo alla costruzione del grafico, l'utente puó premere il pulsante di submit. La pressione di tale bottone effettua una chiamata alla funzione *drawChart()*. Essa ha, prima di tutto, il compito di controllare che i campi riempiti nel form siano corretti dal punto di vista sintattico. Dopo di che, se la verifica è andata a buon fine, recupera dal frame contenente la mappa geografica la struttura dati nella quale sono memorizzate le singole osservazioni (nel seguito verrá descritta la pagina in questione). In questo modo, si evita di dover effettuare piú volte una richiesta di GetObservation verso il modulo SOS: questa richiesta viene infatti effettuata una volta per tutte all'interno del frame che contiene la pagina con la mappa geografica. Basta quindi ottenere un riferimento alla pagina contenente tale

struttura dati per poter svolgere operazioni future che richiedano la conoscenza dei valori numerici delle singole osservazioni. Una volta ottenuto il riferimento alla struttura, il passo successivo è la costruzione del grafico. Vengono aggiunte quindi le due colonne della tabella che conterrá i dati da inserire all'interno del grafico. In particolare si crea una colonna per contenere l'informazione sulla data e una colonna per contenere l'informazione sul valore del parametro RSSI. Una volta costruita la tabella, la funzione comincia a scorrere le singole osservazioni per verificare quali di esse ricadono all'interno dell'area geografica specificata. Per fare questo è sufficiente recuperare da ogni osservazione il valore sulla latitudine e la longitudine ed inserirlo nella formula del calcolo della distanza tra due punti espressi come coordinate geografiche. La formula utilizzata all'interno della funzione è la seguente:

*distanza = acos (sen(lat1) * sen(lat2) + cos(lat1) * cos(lat2) * cos(long1-long2)) * Rt*

Dove:

- **lat1** rappresenta la latitudine inserita nel form.
- **lat2** rappresenta la latitudine dell'osservazione che si sta verificando.
- **long1** rappresenta la longitudine inserita nel form.
- **long2** rappresenta la longitudine dell'osservazione che si sta verificando.
- **Rt** rappresenta il valore del raggio terrestre.

La distanza viene calcolata per ogni osservazione contenuta nella struttura dati. Ogni volta che si ottiene un risultato si va a controllare che questo sia minore del valore del raggio espresso in metri inserito nel form. Se questo avviene, l'osservazione viene inserita all'interno della tabella e il suo contributo verrá inserito nel grafico scatter. Per inserire un elemento all'interno della tabella, la funzione offerta dalle API di Google è la seguente:

dataTable.addRow([new Date(aaaa,mm,gg,hh,m,ss) , vect[i].rssi / vect[i].copies]);

Il passo successivo è quello di richiamare una ulteriore funzione, offerta dalle API di Google, per ordinare i valori inseriti all'interno della tabella in ordine cronologico. A questo punto è sufficiente richiamare la altra funzione che consente di mostrare a video il

grafico appena creato: la funzione *draw()*. Questa, richiede come parametri la tabella con i valori delle osservazioni coinvolte e una variabile contenente opzioni sulla sua visualizzazione. Occorre infine fare una piccola precisione: le coordinate da inserire all'interno del form rappresentano il centro di un'area circolare all'interno della mappa. L'utente, nella scelta della latitudine e della longitudine di tale punto, potrebbe trovarsi in difficoltá; per risolvere questo problema è stata inserita una funzione che, nel momento in cui si effettua un click all'interno della mappa, recupera le coordinate di tale punto e le inserisce in maniera automatica all'interno del form. Si rimanda pertanto all'utente solamente la decisione sul raggio che specifica la dimensione dell'area circolare (che per default viene posto a 1000 metri).

- *deleteChart():* questa funzione non offre nessun servizio all'utente, ma semplicemente consente di nascondere sia il grafico che in quel momento viene mostrato nella pagina che l'area circolare disegnata all'interno della mappa.

In Figura 6.3 viene mostrato lo screenshot della pagina dedicata alla gestione dei grafici.

4.1.9 RssiMap.html

La pagina RssiMap.html è l'ultima delle tre pagine che vengono caricate all'interno dei frames messi a disposizione da Index.html. In questa pagina, l'utente puó visualizzare in una mappa geografica tutte le osservazioni prodotte dai sensori registrati. Come la pagina precedentemente descritta, anche RssiMap.html contiene codice HTML e Javascript ed anche in questo caso la presenza di quest'ultimo è forzata dal fatto che, per la gestione della mappa geografica, si è scelto di utilizzare le API di Google. Queste, rispetto ad una soluzione come le Static API, sempre offerte da Google, offrono moltissime funzioni in piú per la manipolazione degli elementi di una mappa e inoltre, come giá detto, non presentano il limite sul numero dei caratteri presenti nelle richieste HTTP di tipo GET (per mezzo delle quali si basa il principio di funzionamento delle Static API). All'interno di RssiMap.html si trovano sia la struttura dati che ospita tutte le osservazioni che le funzioni necessarie alla gestione della mappa.

Struttura dati per le osservazioni

Come giá detto in precedenza, le osservazioni prodotte dai sensori vengono inserite all'interno di una struttura dati definita in fase di caricamento della pagina. Questo fatto costituisce un forte vantaggio per l'applicazione web; infatti è necessario inoltrare la richiesta di GetObservations verso il modulo SOS solamente una volta durante tutta la durata dell'applicazione. Tale richiesta viene infatti effettuata al primo caricamento o ad ogni refresh; in questo modo le osservazioni inserite all'interno della mappa geografica sono quelle effettivamente presenti all'interno del database gestito dal modulo SOS, in quel determinato momento. Se mentre si sta utilizzando RSSIWatcher vengono iniziati nuovi task, le nuove osservazioni non saranno mostrate a video; per visualizzarle occorre quindi effettuare il refresh della pagina. La struttura dati nella quale vengono inserite le osservazioni viene costruita inserendo al suo interno i seguenti campi:

- **sensorId:** contiene l'identificativo del sensore che ha prodotto l'osservazione. Si ricorda che tale identificativo è rappresentato dal numero di telefono all'interno del quale si trova il sensore che ha prodotto l'osservazione.

- **numObs:** all'interno di questo campo viene memorizzato il numero delle osservazioni totali prodotte da tale sensore. Questo dato puó essere particolarmente utile all'utente che sta analizzando l'osservazione, in modo da farsi un idea del grado di partecipazione del sensore nelle campagne di task realizzate fino a quel momento.

- **timestamp:** costituisce la marca temporale che identifica il momento in cui l'osservazione in questione è stata prodotta.

- **latitudine:** rappresenta la componente di latitudine che indica, insieme alla componente di longitudine, in quale punto del territorio geografico è stata prodotta l'osservazione.

- **longitudine:** rappresenta la componente di longitudine che indica, insieme alla componente di latitudine, in quale punto del territorio geografico è stata prodotta l'osservazione.

- **cellId:** indica la cella con la quale era collegato lo smartphone nel momento in cui ha prodotto tale osservazione.

- **rssi:** indica il parametro fondamentale per questo lavoro di tesi, ovvero la potenza del segnale misurata in quella determinata osservazione.

- **accuracy:** rappresenta il raggio di incertezza sulla misurazione delle coordinate geografiche effettata dallo smartphone. Piú questo numero è piccolo piú la rilevazione delle coordinate geografiche è stata precisa.

- **copies:** questo numero viene incrementato ogni volta che, per le solite coordinate geografiche, viene effettuata una nuova osservazione. Nel caso in cui le coordinate geografiche non cambino tra un periodo di misurazione e l'altro, all'interno del database gestito da SOS sono presenti piú osservazioni distinte ma aventi le solite coordinate geografiche. In questo caso il campo copies contiene il numero di osservazioni effettuate su queste coordinate geografiche e se questo numero è maggiore di 1 il campo rssi contiene la somma delle varie misurazioni su tali coordinate. All'utente di RSSIWatcher viene mostrata una media ottenuta dalla divizione tra il campo rssi ed il campo copies (la stessa media viene calcolata per il campo accuracy descritto al punto precedente).

Metodi offerti

Oltre all'importantissima struttura dati che tiene memoria di tutte le osservazioni prodotte dai sensori, all'interno di questa pagina si trovano anche tre funzioni:

- *initialize():* funzione javascript invocata ogni volta che la pagina viene caricata. È la funzione piú importante presente all'interno della pagina, in quanto si preoccupa di inizializzare in maniera opportuna la struttura dati descritta in precedenza inserendo i dati presenti all'interno delle osservazioni. Il primo passo di tale funzione è quello di effettuare una richiesta con la tecnica AJAX verso la servlet GetObs.jsp (descritta in precedenza nella sezione riguardante i Beans). Si ricorda che tale servlet non faceva altro che effettuare una richiesta di GetObservation al modulo SOS e restituire al richiedente una stringa contenente le varie osservazioni opportunamente separate da particolari caratteri. Alla fine della richiesta, la funzione va a isolare i dati delle singole osservazioni inserendole volta volta all'interno della struttura dati dedicata. In particolare la funzione controlla che, per ogni osservazione ricevuta,

le coordinate siano distinte dalle precedenti; nel caso in cui ci siano due osservazioni diverse con le solite coordinate, viene effettuata la somma del valore del parametro rssi e dell'accuratezza, dopo di che viene incrementato il parametro copies per consentire il corretto calcolo della media. Dopo aver terminato di costruire la struttura dati, la funzione effettua una chiamata ad un altro metodo, il quale si preoccupa di disegnare la mappa geografica.

- *printMap():* questa funzione viene invocata al termine del caricamento delle osservazioni nella struttura dati. Per prima cosa viene caricata la mappa all'interno dell'apposito DIV; dopo di che si fa uso della struttura dati delle osservazioni che, scorrendola completamente, permette di controllare i valori di rssi in ognuna di esse e di vedere a quale categoria appartengono. Le categorie scelte sono 5: la prima comprende tutte le osservazioni che hanno valore inferiore o uguale a 8 sul parametro rssi, la seconda comprende i livelli tra il 9 e il 13, la terza tra il 14 e il 15, la quarta tra il 16 e il 17 e l'ultima comprende i valori maggiori di 17. Ad ogni livello corrisponde una particolare icona colorata con un colore che puó andare dal rosso (segnale debole) al verde intenso (segnale perfetto); in questa fase viene costruito il marker da associare all'osservazione, andando a decidere l'icona e la descrizione da abbinare (in precedenza si sono descritte le due tipologie di descrizioni e di icone). Vengono immediatamente creati tutti e due i tipi di marker, ma solo in uno di essi viene settato la mappa visualizzata a video, mentre l'altra tipologia resta nascosta fino al momento in cui l'utente non decide esplicitamente di effettuare lo scambio di icone. All'interno di questa funzione viene inserito inoltre il listener, il quale si mette in ascolto dei click effettuati dall'utente all'interno della mappa, in modo da prelevare le coordinate geografiche e di memorizzarle all'interno del FORM dedicato alla creazione dei grafici. Alla tipologia di marker utilizzata per avere una visione piú dettagliata delle osservazioni, viene inoltre inserito un listener per l'evento *mouseover*: grazie a questo listener, quando l'utente passa il cursore del mouse sopra il marker, viene disegnata un area circolare che ha per centro la posizione del marker e come raggio il valore dell'accuratezza sulle coordinate geografiche. Si puó assumere, quindi, che il marker possa trovarsi in qualsiasi punto all'interno dell'area disegnata. Nella parte finale di questo metodo viene richiamata la funzione che si preoccupa di costruire il bottone da posizionare in alto sulla mappa, il quale permette di effettuare il passaggio da una tipologia all'altra

dei marker.

- *HomeControl(homeControlDiv, map):* questa funzione richiede come parametri un riferimento alla mappa in questione e un elemento DIV creato appositamente per rappresentare il bottone da premere per effettuare lo scambio di icone. All'interno di questa funzione non si fa altro che gestire lo stile del nuovo DIV inserito e associare un listener che attende che l'utente vi effettui un click. Quando questo accade, la funzione scorre l'array di marker attualmente visualizzati sulla mappa andando a togliere per ognuno di essi il riferimento alla mappa visualizzata. Dopo di che scorre l'array contenente l'altra tipologia di marker andando a settare la mappa (precedentemente impostata al valore null) in modo da renderli visibili.

In Figura 6.2 viene mostrato lo screenshot della pagina dedicata alla gestione della mappa. In questa immagine, si puó notare che è presente una osservazione (in basso a sinistra, di colore rosso) la quale, per poco, non rientra all'interno dell'area geografica sulla quale verrá costruito il grafico scatter. Grazie all'inserimento del valore dell'accuratezza all'interno della formula del calcolo della distanza, tale osservazione andrá comunque a fornire il suo contributo all'interno del grafico: infatti, come si nota dall'immagine, l'intersezione tra le circonferenze disegnate (quella relativa all'area geografica per costruire il grafico e quella relativa all'incertezza della misurazione sulle coordinate) non è un insieme vuoto, ma hanno un area a comune.

4.1.10 GetObs.jsp

Questa pagina è costituita solamente da codice Java e quindi rappresenta in tutto e per tutto una Servlet che svolge dei servizi a chi effettua una richiesta, senza il bisogno di mostrare a video nessun tipo di risultato. Viene utilizzata dalla pagina RssiMap.html per recuperare, in una forma facilmente leggibile, l'insieme delle osservazioni prodotte dai sensori registrati. Per fare questo è necessario creare un istanza del Bean ConnectionUtils e invocare la funzione *getCapabilities* per effettuare preventivamente il recupero degli identificativi dei sensori registrati. Fatto questo, viene effettuata una nuova chiamata alla funzione *getObs*, passando come parametro l'array degli id appena creato. In questo modo si riceve come risposta un documento XML contenente tutte le informazioni contenute all'interno del database gestito dal modulo SOS. Si ricorda che

la funzione getObs del Bean ConnectionUtils non ritorna direttamente il file XML ma ne effettua preventivamente una elaborazione per rendere piú semplice la costruzione della struttura dati che ospita le osservazioni.

4.2 CSAMS

Oltre a modificare il file XML di configurazione di CSAMS, inserendo parametri caratterizzanti il nuovo sensore, si sono rese necessarie anche inoltre delle modifiche all'interno del codice dell'applicazione stessa. Il problema nasce dal fatto che, nella sua versione iniziale, CSAMS gestiva solamente tipologie di sensori che non servivano in questo lavoro di tesi. Il nuovo sensore è stato giá descritto nella sezione dedicata all'architettura e quindi in questa sezione si forniscono solamente i dettagli implementativi, che descrivono con un maggiore livello di dettaglio il comportamento dei moduli modificati o creati ex-novo per rendere funzionante il nuovo sensore.

4.2.1 RSSIWrapper.java

Si ricorda che ogni sensore gestito da CSAMS ha una propria componente software che costituisce l'interfaccia tra l'hardware del dispositivo mobile e il resto dello stack CSAMS. Esso traduce le nuove richieste di nuovi task in chiamate alle funzioni di libreria (quest'ultime regolano il funzionamento dello specifico sensore). RSSIWrapper estende la classe *CSAMSWrapper* ed al suo interno si trovano i seguenti metodi:

- *public RSSIWrapper(int sensorIndex, Context ctx):* rappresenta il costruttore della classe RSSIWrapper e richiede come parametri il numero intero che identifica il sensore dedicato al servizio di monitoraggio del parametro RSSI (in questo caso il numero 12) e il contesto attuale. All'interno del costruttore si trova semplicemente una chiamata al costruttore della superclasse CSAMSWrapper.

- *public SOSObservation processObservation(float[] values):* questa funzione viene chiamata nel momento in cui si hanno a disposizione tutti i dati che compongono una osservazione. Quello che avviene all'interno di essa è effettuare una chiamata al costruttore della classe *RSSIObservation*, che rappresenta la struttura dati che ospita i valori dell'osservazione (in seguito verranno forniti ulteriori dettagli riguardo il ruolo che svolge questa classe). L'istanza del-

la classe appena creata viene ritornata al termine della funzione processObservation.

All'interno di RSSIWrapper.java si trova inoltre una ulteriore classe:

- **RSSIObservation:** questa estende la classe *SOSObservation*, la quale non fa altro che associare la marca temporale all'insieme dei valori che costituiscono l'osservazione stessa. RSSIObservation deve fornire l'implementazione del metodo astratto *toStringFormat()*, il quale deve essere quindi diverso per ogni tipologia di sensore che si vuole gestire; nel caso di questo lavoro di tesi non deve far altro che restituire sotto forma di stringa i parametri che caratterizzano l'osservazione in questione.

4.2.2 CSAMSWrapper.java

Questa particolare classe costituisce il thread che viene lanciato ogni qualvolta si presenti una nuova richiesta di task. All'interno di questo thread si richiama la funzione che gestisce la richiesta per il sensore coinvolto. Per questo lavoro di tesi si è resa necessaria una modifica a questo file: in particolare si è dovuta creare la nuova funzione da richiamare nel caso in cui sia presente una richiesta di task da parte del sensore appena creato. Il metodo richiamato per la gestione del nuovo sensore è il seguente:

- *void executeRSSITask():* questa è la funzione che svolge il ruolo fondamentale nel recupero delle informazioni lette dai sensori presenti all'interno dello smartphone. Tale funzione è fatta in modo da essere richiamata una volta sola per ogni nuovo task; pertanto, al suo interno si è dovuto pensare a un modo per gestire sia il prelievo dei dati dai sensori, sia il ciclo di attesa che rappresenta il periodo di campionamento del task in questione. Dopo aver inizializzato correttamente le componenti responsabili della gestione dei sensori coinvolti, tale metodo gestisce il richiamo periodico alle chiamate delle funzioni che gestiscono i sensori fisici. Tale periodicitá viene ottenuta grazie all'elemento *Handler* offerto da Android: si tratta di una componente da utilizzare nel momento in cui si ha intenzione, all'interno di un thread, di eseguire qualche operazione dopo un certo periodo di tempo (in questo caso allo scadere del periodo di campionamento). Tale meccanismo si basa sull'invio di messaggi all'interno del thread che saranno catturati dalla funzione

handleMessage. All'interno di questo metodo, richiamato ogni volta che un messaggio viene inviato, vengono dunque effettuate le chiamate alle funzioni di libreria che gestiscono il funzionamento dei sensori. I messaggi inviati a tale Handler possono essere spediti dopo un certo intervallo di tempo impostabile a piacimento; in questo caso, l'intervallo di tempo deve essere pari al periodo di campionamento del task. Ricapitolando, ogni nuovo messaggio viene inviato in coincidenza dello scadere di tutti i periodi di campionamento e quindi la funzione handleMessage viene invocata in quegli istanti precisif. I messaggi inviati possono essere di due tipi diversi, ognuno progettato per svolgere una funzione differente:

- **Tipo 1:** i messaggi del primo tipo costituiscono lo scadere dei periodi di misurazione e quindi, quando viene ricevuto un messaggio di questo tipo, vengono richiamate tutte le funzioni che gestiscono i sensori in modo da recuperare i parametri misurati. Dopo aver prelevato i dati dai sensori, si controlla che il messaggio appena ricevuto sia uno intermedio o che sia l'ultimo del task; nel primo caso, viene di nuovo preparato un messaggio del primo tipo e viene inviato con un ritarto pari al periodo di campionamento, mentre nel secondo caso viene creato e inviato immediatamente un messaggio del tipo 0.
- **Tipo 0:** un messaggio del tipo 0 rappresenta il messaggio da inviare per indicare all'applicazione che il task in esecuzione ha terminato la sua attivitá. In questo caso, quando viene ricevuto tale messaggio dall'handler, si vanno a rimuovere i vari listener e viene chiuso il ciclo di chiamate alle funzioni dei sensori.

4.2.3 CSAMSRssiLocationListener.java

Questa particolare classe è stata costruita ex-novo per la gestione della componente GPS del nuovo sensore. Come si intuisce dal nome, la classe implementa il *LocationListener* in modo da poter rimanere in ascolto di eventuali aggiornamenti sulle coordinate geografiche. I parametri che tale listener deve monitorare sono tre: la latitudine, la longitudine e l'accuratezza della misurazione delle due precedenti. Il listener è stato costruito in modo da effettuare semplicemente la memorizzazione dei tre parametri appena elencati all'interno di tre variabili. Per fare questo è bastato ridefinire la funzione *onLocationChanged()*, la quale viene richiamata ogni volta che lo smartphone sperimenta una variazione sulla po-

sizione nel territorio geografico. Inoltre, all'interno di questa classe, si rendono disponibili all'applicazione altri metodi, i quali sono quelli effettivamente chiamati allo scadere dei vari periodi di campionamento. Queste funzioni vanno semplicemente a prelevare le ultime informazioni memorizzate riguardanti la misurazione sulle coordinate geografiche e le restituiscono immediatamente. Questo garantisce che, allo scadere di un periodo di campionamento, siano sempre presenti dei valori aggiornati sulla posizione dello smartphone.

4.2.4 CSAMSRssiSignalStrenghtListener.java

Analogamente al caso precedente, si è deciso di costruire ex-novo anche un listener capace di rimanere in ascolto di eventuali cambiamenti sulla potenza del segnale ricevuto. Per effettuare questa operazione è bastato estendere *PhoneStateListener* ed implementare il metodo *onSignalStrenghtsChanged()*, il quale viene richiamato ogni volta che il telefono sperimenta una variazione sul parametro RSSI. All'interno di questo metodo, come nel caso di *CSAMSRssiLocationListener*, non si fa altro che tenere in memoria il nuovo valore misurato. All'interno di questa classe si trova anche una funzione che viene richiamata ogni volta che scade un periodo di misurazione del task in esecuzione: questo metodo non fa altro che prelevare l'ultima informazione memorizzata nella variabile che tiene traccia dell'andamento del segnale RSSI e la restituisce al chiamante. In questo modo si garantisce che, allo scadere del periodo di campionamento del task, sia sempre presente una informazione valida sul valore del parametro RSSI.

4.2.5 CSAMSRssiManager.java

Per terminare l'elenco delle componenti modificate o create in CSAMS durante questo lavoro di tesi, occorre introdurre questa ultima classe. All'interno di essa, si memorizzano i dati relativi al task in arrivo (durata, periodo di campionamento e di conseguenza i cicli necessari al completamento del task). Quando il modulo CSAMSWrapper ha completato il prelievo dei dati dai sensori coinvolti, effettua una chiamata all'unico metodo offerto all'interno di questa nuova classe:

- *public int sendInfo(float rssiValues[]):* questo metodo richiede come parametro l'array contenente i dati prelevati dai sensori e si preoccupa di effettuare l'invio di essi verso la componente che li inoltrerá

al modulo SOS; il tutto accade in maniera condizionata a seconda che le informazioni sulla posizione siano correttamente aggiornate. Un'altra operazione fondamentale che svolge questo metodo è quella di regolare il ciclo di osservazioni: infatti, al suo interno, si tiene memoria dei cicli che mancano alla terminazione del task. Ogni volta che il CSAMSWrapper chiama il metodo sendInfo, questo restituisce un valore intero che indica se il task sia completo o meno; tale valore viene utilizzato per definire quale tipologia di messaggio deve gestire l'Handler (valore 1 significa altre osservazioni rimanenti, mentre valore 0 significa che il task è terminato).

Capitolo 5

Pubblicazione del servizio

Dopo aver descritto in linea generale i compiti che devono svolgere i singoli moduli e dopo aver descritto quest'ultimi con dettagli implementativi, non resta che andare a effettuare una esauriente campagna di acquisizione dei dati. All'inizio di questo processo ci si è trovati peró di fronte ad un problema fondamentale: si trattava di rendere raggiungibili i terminali mobili da parte del lato server, andando a sfruttare la connessione dati offerta dal proprio gestore telefonico.

5.1 Il problema

In questa sezione si descrivono le varie motivazioni che hanno spinto alla decisione di rendere l'intero servizio totalmente disponibile e funzionante all'interno della rete internet. Una volta finita di creare l'applicazione web RSSIWatcher e una volta effettuate le opportune modifiche a CSAMS in modo da consentire la gestione del nuovo sensore introdotto, si è passati alla raccolta dei dati. Inizialmente, le prime prove sono state svolte all'interno di una rete WiFi; quindi ci si trovava di fronte ad una situazione in cui sia il server Tomcat che il dispositivo mobile erano collegati al medesimo router e tutto funzionava correttamente. In questa panoramica, infatti, non vi era nessun tipo di problema di visibilitá ne da parte del server, ne da parte dello smartphone: le comunicazioni avvenivano dunque senza problemi e sia la fase di registrazione del sensore, sia l'esecuzione dei task che quest'ultimo richiede, avvenivano con successo. Le due parti, per "raggiungersi" a vicenda, utilizzavano gli indirizzi IP privati a loro assegnati all'interno della rete locale (non si presentava quindi nessun tipo di ostacolo tra essi). Non aveva molto senso, peró, effettuare

delle misurazioni esclusivamente all'interno di una abitazione. Prima di questo lavoro di tesi, il sistema sul quale si doveva lavorare, fornito come risultato del lavoro di vecchi laureandi, è stato pensato in modo da rendere disponibile un servizio di acquisizione dati all'interno di una vasta area geografica, teoricamente di copertura planetaria. Tale sistema, nella sua versione primitiva, non è stato peró progettato in questo modo: infatti, andando a disattivare l'antenna WiFi del terminale mobile, viene attivata, se presente, la connessione dati offerta dal proprio gestore telefonico. Il passaggio da un IP privato a un IP pubblico, da parte dello smartphone, ha comportato l'inevitabile problema di visibilitá tra i due indirizzi IP: l'indirizzo pubblico assegnato ad un terminale mobile, da parte del proprio gestore telefonico, si trovava dietro una tabella di NAT (Network Address Traslation) e le richieste che partivano dal server verso lo smartphone non venivano recapitate correttamente.

5.2 La soluzione: Cloud to Device Messaging

Solitamente, l'ostacolo causato dall'utilizzo del Network Address Traslation viene risolto andando a mettere mano sulle impostazioni del router al quale si è collegati. Non potendo risolvere il problema in questo modo, si è dovuto ripensare al funzionamento dell'intero sistema e cambiare totalmente il paradigma di comunicazione nelle connessioni server-smartphone. Una volta reso visibile il server Tomcat all'interno della rete Internet, infatti, l'unico problema da risolvere è costituito dal corretto inoltro delle richieste di nuovi task, le quali viaggiano dal lato server verso le varie applicazioni Android. È bene ricordare che CSAMS, nella sua versione iniziale, dopo aver completato la fase di Boot, si poneva in ascolto su un socket; tale socket veniva aperto sul proprio indirizzo IP pubblico ed era raggiungibile attraverso la porta 8081 (di default questa era la porta scelta da CSAMS). La scelta di questa tecnica presentava inevitabilmente il problema giá esposto; la soluzione ideale, quindi, non dovrebbe prevedere il coinvolgimento degli indirizzi IP. In questo, modo si otterrebbe un sistema il cui funzionamento non sia basato su un parametro che puó cambiare in maniera dinamica e imprevista e che si trova dietro un ostacolo come il NAT. L'ideale sarebbe sostituire quindi l'indirizzo IP con un parametro piú "stabile" e che permetta al terminale mobile di essere sempre raggiungibile, indipendentemente dal valore che assume l'IP pubblico o dalla frequenza con il quale questo cambia. La soluzione a questo problema si chiama *C2DM* (Cloud To Device Messaging). Questo servizio permette, agli sviluppatori di applicazioni Android,

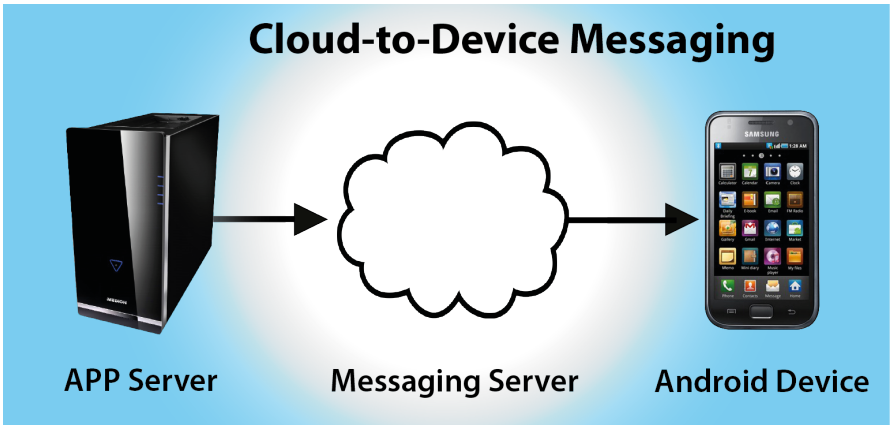

Figura 5.1: Cloud to Device Messaging

di inviare dei dati da una applicazione server verso una applicazione presente sul terminale mobile. Il servizio C2DM gestisce tutti gli aspetti relativi alla gestione delle code di messaggi inviati e la consegna di questi verso una precisa applicazione mobile. Per usufruire di questo servizio offerto da Google, è necessario preliminarmente registrare un nuovo indirizzo mail da utilizzare all'interno di alcuni messaggi.

5.2.1 Introduzione

In seguito, vengono descritte brevemente quali sono i principali compiti e benefici di Android Cloud to Device Messaging:

- Permette alle applicazioni server di inviare messaggi "leggeri" alle applicazioni Android. Il servizio di messaging non è stato progettato per consegnare messaggi contenenti una grossa quantitá di dati, per questo motivo vengono definiti "leggeri". Solitamente, il messaggio spedito verso una applicazione mobile contiene le informazioni necessarie per rendere consapevole l'applicazione stessa che ci sono nuovi dati sul server e che quinidi è necessario aprire una nuova connessione per reperirli. Nel caso di questo lavoro di tesi, l'apertura della seconda connessione non è necessaria, in quanto all'interno del primo messaggio si troverá un informazione che puó essere definita "leggera" del tipo <*operazione-da-gestire, indice-sensore, durata-task, periodo-task*>: una semplice stringa talmente corta da poter essere inserita direttamente nel messaggio stesso.

- C2DM non garantisce ne l'avvenuta consegna dei messaggi inviati alle applicazioni mobili, ne l'ordine di eventuali messaggi spediti simultaneamente.

- L'applicazione Android, grazie a C2DM, adesso non è costretta a rimanere in attesa su un socket. Essa viene "svegliata" dal sistema con la tecnica dei Broadcast Intent quando il messaggio arriva. Basta pertanto registrare correttamente i vari Broadcast Receivers con i relativi permessi, e il gioco è fatto.

- Non viene fornita nessuna interfaccia di gestione dei messaggi. C2DM passa semplicemente i messaggi ricevuti direttamente all'applicazione obiettivo. L'applicazione deve pertanto definire il comportamento da assumere all'arrivo di un nuovo messaggio e questo compito spetta al programmatore dell'applicazione stessa.

- Richiede che il dispositivo Android sul quale viene installata l'applicazione abbia una versione di Android uguale o superiore alla 2.2; tale dispositivo deve avere inoltre installata l'applicazione del Market al suo interno.

- C2DM sfrutta una connessione attiva nei confronti dei servizi di Google. Questo fatto richiede che vi sia almeno un account Google attivo sullo smartphone in questione.

5.2.2 L'architettura C2DM

Questa sezione fornisce una descrizione su come il servizio C2DM lavora. In seguito vengono introdotti alcuni termini utilizzati all'interno di C2DM, i quali possono essere divisi principalmente in due categorie:

- **Componenti:** rappresentano le entitá fisiche coinvolte nell'espletamento del servizio C2DM.

- **Credenziali:** sono i vari identificativi e i token che vengono utilizzati nelle diverse fasi del servizio, in tale modo si assicura che tutte le parti siano correttamente autenticate e che i messaggi vengano ricevuti dalle applicazioni corrette.

Le componenti

Le entitá coinvolte nel servizio di Cloud to Device Messaging sono esattamente tre:

- **Mobile Device:** rappresenta il dispositivo mobile sul quale è installata l'applicazione CSAMS e che usa il C2DM. Si ricorda che questo deve avere il requisito di accogliere una versione maggiore o uguale alla 2.2 e che deve avere almeno un account Goggle attualmente attivo.

- **Third-Party Application Server:** rappresenta una applicazione server che lo sviluppatore deve progettare in maniera opportuna per implementare il servizio di C2DM. Questa è la componente che deve inviare il messaggio verso l'applicazione mobile desiderata, grazie all'ausilio dei server di Google utilizzati per il servizio di C2DM. Nel caso di questo lavoro di tesi, il ruolo di tale componente viene splittato in due componenti giá presenti nel sistema CloudSensor: il Naming Server e l'Android Plugin. Tali componenti sono state pertanto modificate per essere in grado di sostenere correttamente il servizio di C2DM.

- **C2DM Servers:** sono i servers di Google coinvolti nell'inoltro dei messaggi verso i dispositivi Android, da parte delle Third-Application Server.

Le credenziali

Le credenziali si ricordano essere le componenti che garantiscono il corretto funzionamento del sistema, sia in termini di autenticazione da parte delle componenti esposte al punto precedente, sia in termini di rilascio dei messaggi per garantire la raggiungibilitá dell'applicazione di destinazione. Tali credenziali sono esattamente cinque:

- **Sender ID:** Questo identificativo è costituito dall'indirizzo mail associato all'applicazione che si sta sviluppando. Il Sender ID viene coinvolto nel processo di registrazione per abilitare l'applicazione stessa al servizio, in modo da renderla visibile all'interno del sistema C2DM. Solitamente, si cerca di inserire il nome dell'applicazione all'interno dell'indirizzo mail. Nel caso di questo lavoro di tesi è stato creato un nuovo account Google dedicato all'applicazione CSAMS: *appcsams@gmail.com* con password *appcsams*.

- **Application ID:** Rappresenta l'applicazione mobile registrata al servizio C2DM, la quale sará quindi destinata a ricevere messaggi sotto forma di Broadcast Intent da parte dei server di Google. Ogni applicazione viene identificata dal proprio package inserito

come parametro anche nel file Manifest dell'applicazione. Questo assicura che il messaggio venga inoltrato in maniera corretta verso l'applicazione obiettivo.

- **Registration ID:** Questo identificativo assume un ruolo fondamentale per il corretto funzionamento del sistema. Viene rilasciato dai server di C2DM all'applcazione Android che è stata autorizzata a ricevere i messaggi. Una volta che una applicazione ha ricevuto il suo registration ID, deve obbligatoriamente inoltrarlo all'applicazione server che si occupa di inviare i messaggi verso di essa. Tale ID, pertanto, identifica una particolare istanza dell'applicazione su un determinato dispositivo mobile. L'applicazione server utilizzerá questo identificativo come se fosse l'indirizzo pubblico del terminale, per spedirgli i vari messaggi.

- **Google User Account:** Per poter funzionare correttamente, il dispositivo mobile deve includere almeno un account Google attivo all'interno di esso e questa credenziale identifica proprio tale account.

- **Sender Authentication Token:** Questo elemento è un token che deve essere memorizzato all'interno dell'applicazione server. L'Authentication Token fornisce l'autorizzazione necessaria all'applicazione server per poter comunicare con i server di Google e quindi per poter inoltrare loro i messaggi di C2DM. Tale token viene incluso all'interno dell'Header della richiesta POST.

5.2.3 Le fasi di C2DM

Di seguito vengono descritti i processi principali che compongono il servizio di Cloud to Device Messaging:

- **Abilitazione di C2DM:** una applicazione Android che viene eseguita su uno smartphone deve obbligatoriamente registrarsi al servizio per ricevere messaggi di tipo C2DM.

- **Invio di un messaggio:** l'applicazione server appositamente progettata per usufruire del servizio di C2DM, avrá la necessitá in seguito di inviare un messaggio verso una applicazione mobile.

- **Ricezione di un messaggio:** l'applicazione Android intercetta un particolare messaggio a lei destinato, il quale è stato inviato da una applicazione server che ha necessitá di comunicargli qualcosa.

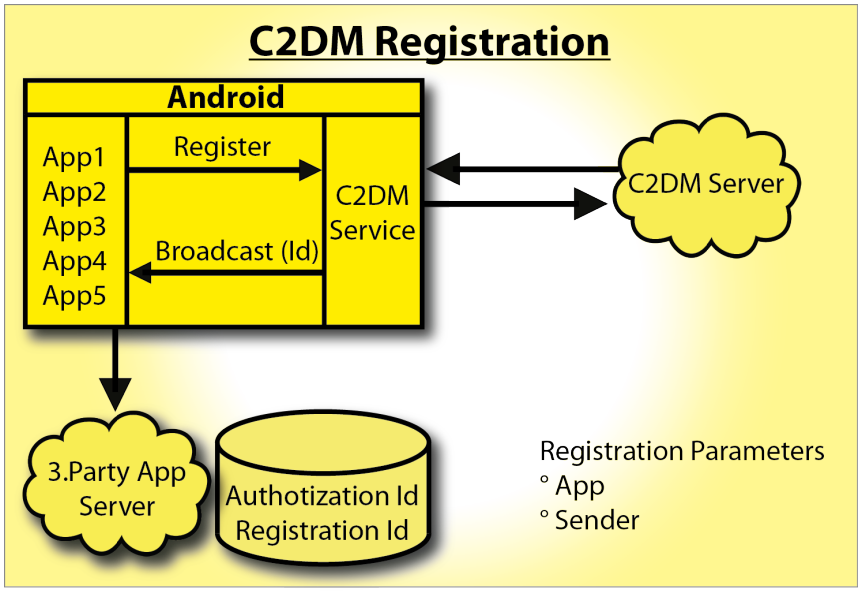

Figura 5.2: Scenario della registrazione a C2DM

Abilitazione di C2DM

Questi sono i singoli passi che deve seguire una applicazione Android per registrarsi al servizio C2DM (vedi anche Figura 5.2):

1. La prima volta che l'applicazione viene avviata deve creare ed inoltrare un Intent verso i server dedicati al C2DM. L'Intent di registrazione *com. google. android. c2dm. intent. REGISTER* include sia il Sender ID (il quale rappresenta l'account autorizzato a inviare messaggi all'applicazione e nel caso di questo lavoro di tesi si tratta di *appcsams@gmail.com*), che l'Application ID (il quale si ricorda essere costituito dal nome del package che rappresenta in modo univoco l'applicazione).

2. Se la registrazione va a buon fine, il server di C2DM esegue il broadcast di un REGISTRATION Intent, il quale deve essere intercettato dall'applicazione obiettivo, in quanto contiene una informazione molto preziosa: il Registration ID. L'applicazione deve memorizzare questo importante identificativo per un uso futuro. I server di Google rinfrescano periodicamente questo ID, quindi si deve tenere conto che il REGISTRATION Intent viene inoltrato piú di una volta. L'applicazione deve pertanto essere sempre pronta a rice-

vere tale Intent e memorizzare l'informazione fresca riguardante il Registration ID.

3. Per completare la fase di registrazione al servizio di C2DM, l'applicazione Android deve rendere il lato server a conoscenza del valore del suo Registration ID. Senza di esso, infatti, il lato server non sarebbe in grado di identificare univocamente le singole istanze delle applicazioni Android installata sui vari dispositivi. In questo caso, CSAMS deve quindi inviare immediatamente al Naming Server il Registration ID appena ricevuto; il Naming Server è stato modificato appositamente per memorizzare le informazioni sui vari Registration ID che riceve.

Il Registration ID è un informazione che rimane valida finché l'applicazione Android non richiede esplicitamente la deregistrazione, oppure finché i server di Google non effettuano il refresh di tale identificativo.

Invio di un messaggio

L'applicazione lato server deve essere in qualsiasi momento pronta, con tutte le credenziali necessarie, ad inviare un messaggio C2DM verso una precisa applicazione Android; per fare questo, i seguenti punti devono essere giá stati completati dall'applicazione server stessa prima dell'inoltro di un messaggio:

- L'applicazione server deve giá essere a conoscenza del Registration ID ricevuto in fase di registrazione dall'applicazione mobile.

- L'applicazione server deve avere giá memorizzato, in una opportuna struttura dati, l'informazione su tale Registration ID; tale parametro, per la corretta ricezione del messaggio da parte del dispositivo mobile, deve essere fresca.

Oltre ai due punti precedenti, l'applicazione server deve essere in possesso anche di un altra importante informazione: il ClientLogin Authorization Token. Tale informazione deve essere in possesso dall'applicazione server giá al suo avvio; si puó immaginare che la fase di richiesta e ricezione del token sia come una specie di "fase di Boot". Il Token serve all'applicazione server per poter mandare messaggi di tipo C2DM verso i server di Google. È sufficiente essere in possesso di un solo Token, il quale puó essere utilizzato per spedire messaggi verso tutte le applicazioni Android che abbiano mandato in precedenza il proprio Registration ID. Quindi,

all'interno dell'applicazione server, si trovano un solo token e molteplici registration ID, uno per ogni applicazione che supporta il servizio di messaging. Ecco la sequenza di eventi che compongono l'invio di un messaggio dall'applicazione server verso un dispositivo Android:

1. L'applicazione server invia il messaggio verso i server di Google.

2. I server di Google accodano e memorizzano tali messaggi nel caso in cui il dispositivo, sul quale è installata l'applicazione destinataria, sia temporaneamente inattivo.

3. Quando un dispositivo diviene attivo, i server di Google inviano il messaggio in attesa di essere spedito.

4. Sul dispositivo mobile, si assiste all'inoltro in broadcast del messaggio che puó essere gestito solo dalla specifica applicazione avente il Broadcast Receiver correttamente registrato. In altre parole, solo l'applicazione obiettivo è in grado di intercettare il messaggio; questo evento "sveglia" l'applicazione stessa, che in quel momento si era posta in attesa di nuove richieste.

5. Dopo aver intercettato l'Intent basterá pertanto splittare in maniera opportuna la stringa che costituisce il campo "payload" dell'Intent stesso, in modo da avere tutte le informazioni necessarie per iniziare un nuovo task.

Ricezione di un messaggio

In seguito si riportano gli eventi che compongono la fase di ricezione di un messaggio su un dispositivo mobile (vedi anche Figura 5.3):

1. Il sistema composto dai server di Google riceve il messaggio in arrivo ed estrae da esso tutte le informazioni sotto forma di *key-value*.

2. Il sistema passa le informazioni estratte dal punto precedente verso l'applicazione Android attraverso l'inoltro di un Intent del tipo *com. google. android. c2dm. intent. RECEIVE*, con un extras contenente i valori estratti dal primo messaggio.

3. L'applicazione Android estrae i dati dall'Intent ricevuto e li processa.

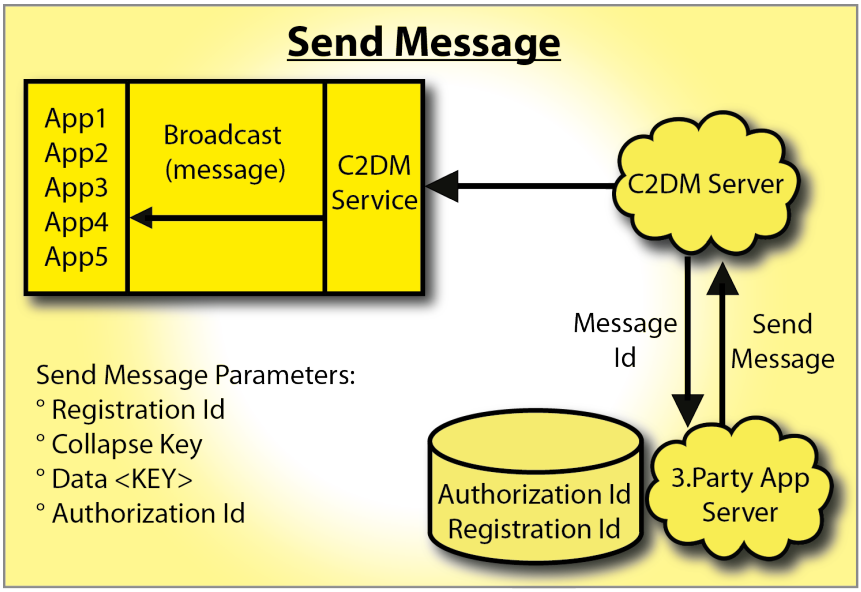

Figura 5.3: Scenario dell'inoltro di un messaggio

Come giá detto in precedenza le informazioni, che dovrebbero essere spedite verso l'applicazione CSAMS, non sono costituite da una ingente quantitá di dati, pertanto si è portata avanti la scelta progettuale di non aprire una seconda connessione verso il lato server per recuperare i parametri del nuovo task. Si è scelto piuttosto di inserire questi parametri direttamente all'interno del campo "payload" dell'Intent che verrá ricevuto dall'applicazione mobile. La stringa in questione viene strutturata come segue: *id-operazione,id-sensore,durata-task,periodo-task*.

A questo punto viene da domandarsi cosa vede un possessore di un particolare smartphone dove si deve installare una applicazione Android che utilizza il servizio C2DM. Quando una applicazione di questo tipo viene installata su di uno smartphone, viene chiesta all'utente l'autorizzazione affinché l'applicazione possa ricevere, via Intent, dei dati attraverso la connessione internet. Una volta accettate le varie condizioni, l'applicazione viene installata correttamente e l'utente non si rende conto di cosa stia succedendo durante tutto il suo ciclo di vita.

5.2.4 Scrivere applicazioni con C2DM

Per scrivere applicazioni Android che utilizzino questo servizio offerto da Google, è necessario dunque aver progettato, tra l'altro, un applicazione server che esegua le operazioni descritte in precedenza. In questo

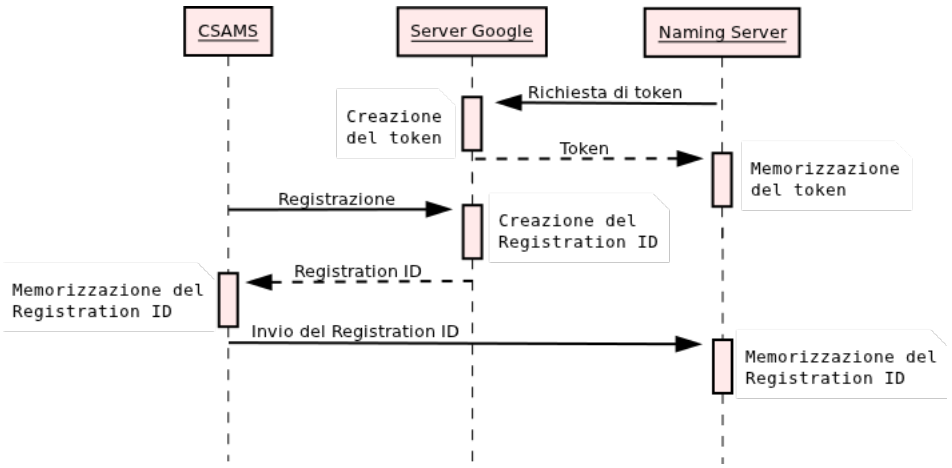

Figura 5.4: Sequenza operazioni per la registrare l'applicazione a C2DM

lavoro di tesi, i moduli Naming Server e Android Plugin sono quelli che, insieme, ricoprono il ruolo dell'applicazione server. In questa sezione, verranno descritti i concetti fondamentali che dovranno essere inseriti in fase di progettazione di una applicazione Android che desidera usufruire del servizio di C2DM. Si premette che non esiste nessuna interfaccia utente associata alla Framework C2DM; i messaggi ricevuti dovranno essere quindi processati all'interno dell'applicazione che si sta sviluppando. Verranno adesso elencati i passi principali da eseguire per costruire questo tipo di applicazioni:

- Creazione di un Manifest File che contenga i dovuti permessi per utilizzare il C2DM.
- Implementare con codice Java componenti che:
  1. permettano di iniziare il servizio di registrazione verso i server di Google,
  2. registrino correttamente i Receivers per gli Intent *com. google. android. c2dm. intent. C2D_MESSAGE* e *com. google. android. c2dm. intent. REGISTRATION*.

Creazione del file Manifest

Ogni applicazione Android deve possedere un proprio Manifest File con estensione XML. Il file manifest presenta al sistema Android delle informazioni relative all'applicazione. Per poter utilizzare il servizio C2DM

è stato necessario pertanto modificare il giá esistente file manifesto di CSAMS ed in particolare si sono dovute inserire le seguenti informazioni:

- Per prima cosa si è inserito il permesso *com. google. android. c2dm. permission. RECEIVE* per indicare che l'applicazione ha l'autorizzazione a registrarsi al servizio e quindi a ricevere i relativi messaggi.

- Il permesso *android.permission.INTERNET* (che comunque era giá presente) per fornire l'autorizzazione a inviare il Registration ID verso l'applicazione server, utilizzando la connessione dati del telefono.

- Il permesso *application-package.permission.C2D_MESSAGE* necessario per evitare che altre applicazioni si possano registrare o possano ricevere messaggi di tipo C2DM non destinati a loro (al posto di *application-package* va sostituito l'effettivo valore del package dell'applicazione in questione).

- Nel file manifesto si devono inoltre registrare in maniera *statica* i due Broadcast Receivers per gli Intents *com. google. android. c2dm. intent. RECEIVE* e *com. google. android. c2dm. intent. REGISTRATION*. A differenza della registrazione dinamica, la quale avviene all'interno del codice, la registrazione statica, che avviene nel file manifesto, permette di registrare dei Receivers in grado di rimanere attivi per tutta la durata dell'applicazione. I due Receivers serviranno pertanto ad intercettare messaggi di registrazione (contenenti il Registration ID) e saranno utili inoltre per ricevere i messaggi C2DM mandati dall'applicazione server.

- Se il servizio C2DM è di fondamentale importanza per il corretto funzionamento dell'applicazione, è necessario inserire nel file manifesto anche una riga di questo tipo *android:minSdkVersion=8*. In questo modo si ha la sicurezza che l'applicazione non venga installata all'interno di dispositivi che non abbiano i requisiti di sistema necessari a supportare il C2DM.

Di seguito viene mostrato un listato contenente un modello di file manifesto per tutte le applicazioni che utilizzano il servizio C2DM:

Listing 5.1: File Manifesto tipico, "AndroidManifest.xml"

```
1 <manifest package="com . example . myapp" ...>
```

```
    <!-- Only this application can receive the messages and registration
        result -->
    <permission android:name="com . example . myapp . permission .
        C2D_MESSAGE" android:protectionLevel="signature" />
    <uses-permission android:name="com . example . myapp . permission .
        C2D_MESSAGE" />

    <!-- This app has permission to register and receive message -->
    <uses-permission android:name="com . google . android . c2dm .
        permission . RECEIVE" />

    <!-- Send the registration id to the server -->
    <uses-permission android:name="android . permission . INTERNET"
        />

    <application...>
      <!-- Only C2DM servers can send messages for the app. If
          permission is not set - any other app can generate it -->
      <receiver android:name=".C2DMReceiver" android:permission="com
          . google . android . c2dm . permission . SEND">
          <!-- Receive the actual message -->
          <intent-filter>
              <action android:name="com . google . android . c2dm . intent
                  . RECEIVE" />
              <category android:name="com . example . myapp" />
          </intent-filter>
          <!-- Receive the registration id -->
          <intent-filter>
              <action android:name="com . google . android . c2dm . intent
                  . REGISTRATION" />
              <category android:name="com . example . myapp" />
          </intent-filter>
      </receiver>
      ...
    </application>
    ...
</manifest>
```

La registrazione a C2DM

Dopo aver visto come modificare in maniera opportuna il file Manifest dell'applicazione, in questa sezione viene descritto come viene effettuata

la registrazione al servizio di C2DM, all'interno della quale si ha il bisogno di andare a comunicare direttamente con i server di Google. Per prima cosa, quello che deve fare l'applicazione è mandare un Intent del tipo *com. google. android .c2dm. intent. REGISTER* contenente due parametri inseriti come extra:

- **sender:** questo campo rappresenta l'ID dell'account autorizzato a inviare messaggi alle applicazioni mobili (si ricorda che in questo lavoro di tesi risulta essere *appcsams@gmail.com*).

- **app:** questo extra contiene l'identificativo dell'applicazione e viene settato con l'ausilio di un PendingIntent, il quale fornisce al servizio di registrazione di estrarre informazioni riguardanti l'applicazione stessa.

Ecco un esempio tipico di codice da inserire all'interno di una applicazione mobile per iniziare il processo di registrazione verso i server di Google:

Listing 5.2: Intent di registrazione, “

```
Intent registrationIntent = new Intent("com . google . android . c2dm .
    intent . REGISTER");
registrationIntent.putExtra("app", PendingIntent.getBroadcast(this, 0, new
    Intent(), 0)); // boilerplate
registrationIntent.putExtra("sender", emailOfSender);
startService(registrationIntent);
```

La fase di registrazione non puó essere definita terminata finché l'applicazione mobile non riceve il Registration ID dai server di C2DM e non lo invia all'applicazione server, che dovrá riutilizzarlo quando avrá bisogno di comunicare con il terminale mobile.

Analogamente alla fase di registrazione, la fase di deregistrazione puó essere iniziata nel seguente modo:

Listing 5.3: Intent di deregistrazione, “

```
Intent unregIntent = new Intent("com . google . android . c2dm . intent .
    UNREGISTER");
unregIntent.putExtra("app", PendingIntent.getBroadcast(this, 0, new Intent
    (), 0));
startService(unregIntent);
```

Gestione del risultato della registrazione

Come accennato in precedenza, quando si è parlato delle modifiche da apportare al file Manifest, occorre definire e registrare in maniera statica due Broadcast Receivers. Solo in questo modo è possibile ricevere gli Intent del tipo *com. google. android. c2dm. intent. REGISTRATION* e del tipo *com. google. android. c2dm. intent. RECEIVE*. In questa sezione si descrivono in dettaglio quali sono le operazioni principali che deve effettuare il ricevitore del primo tipo. Il suo principale ruolo è quello di estrarre, dall'Intent in arrivo, l'informazione riguardante il parametro Registration ID. Va detto che l'Intent di registrazione puó essere spedito dai server di Google in ogni momento: si ricorda, infatti, che questo importante parametro resta valido finché l'applicazione Android non richiede esplicitamente la deregistrazione o finché i server di C2DM non ritengono opportuno rinfrescarlo. In quest'ultimo caso, l'Intent viene spedito nuovamente verso l'applicazione mobile. Il ricevitore in questione deve pertanto essere pronto ad intercettare il messaggio in qualsiasi momento. Dopo aver estratto il valore del parametro Registration ID, il Broadcast Receiver deve immediatamente inoltrarlo all'applicazione server. L'intent REGISTRATION viene quindi generato dai server di Google e al loro interno si trova tra l'altro anche un campo "errore". Se all'interno di questo si trova effettivamente un errore, occorre andare ad estrarre ulteriori informazioni sulle cause che lo hanno generato. Verranno elencati adesso i possibili errori che si possono trovare in questa fase:

- **SERVICE_NOT_AVAIABLE:** il dispositivo non è in grado di leggere la risposta oppure c'è stato un errore del tipo 500 o 503 dal server. Buona norma sarebbe ritentare di inviare nuovamente la richiesta.

- **ACCOUNT_MISSING:** all'interno dello smartphone non è presente un account Google attualmente attivo.

- **AUTHENTICATION_FAILED:** errore sul campo password.

- **TOO_MANY_REGISTRATIONS:** l'utente ha molteplici applicazioni registrate. La soluzione a questo errore è la disinstallazione di alcune di esse.

- **INVALID_SENDER:** l'account sender non viene riconosciuto dal sistema.

- **PHONE_REGISTRATION_ERROR:** il terminale che sta tentando di registrarsi non supporta il servizio di C2DM.

Uno schema classico sul quale viene costruito un Broadcast Receiver di questo tipo è il seguente:

Listing 5.4: Broadcast Receiver, "Gestione della registrazione"

```
public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        Log.w("C2DM", "Registration Receiver called");
        if ("com.google.android.c2dm.intent.REGISTRATION".
            equals(action)) {
                Log.w("C2DM", "Received registration ID");
                final String registrationId = intent
                                .getStringExtra("registration_id");
                String error = intent.getStringExtra("error");

                Log.d("C2DM", "dmControl: registrationId = " +
                    registrationId
                                + ", error = " + error);
                // TODO Send this to my application server
        }
```

Gestione di un messaggio in arrivo

Dopo aver completato correttamente la fase di registrazione al servizio, l'applicazione Android è pronta ad attendere l'arrivo di un messaggio da parte dell'applicazione server. Come giá detto in precedenza, il primo passo in questo processo lo compie l'applicazione server, in quanto è la componente che ha bisogno di comunicare qualcosa all'applicazione mobile. Il messaggio viene innanzitutto inviato ai server di C2DM che lo processano in modo da estrarre le informazioni dal payload, per incapsularle all'interno dell'Intent che deve costruire. Una volta pronto, tale Intent viene finalmente inoltrato all'applicazione obiettivo, la quale deve essere in grado di processare il nuovo messaggio estraendo le informazioni di interesse che sono state inserite sotto forma di "extra". Ecco un esempio di codice per un Broadcast Receiver che ha il compito di gestire l'arrivo di questo tipo di Intent:

Listing 5.5: Broadcast Receiver, "Gestione del messaggio"

```
public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
```

```
3                   Log.w("C2DM", "Message Receiver called");
                    if ("com.google.android.c2dm.intent.RECEIVE".equals(
                        action)) {
5                       Log.w("C2DM", "Received message");
                        final String payload = intent.getStringExtra("
                            payload");
7                       Log.d("C2DM", "dmControl: payload = " +
                            payload);
                        // Send this to my application server
9                   }
            }
```

In Figura 5.4 viene mostrata in un diagramma la sequenza di operazioni effettuate in questa fase.

5.2.5 Il ruolo dell'applicazione server

L'applicazione server, che provvederá a inviare messaggi verso le applicazioni Android correttamente registrate, ha bisogno di soddisfare i seguenti requisiti:

- Deve essere capace di comunicare con i clienti.
- Deve essere capace di inviare richieste di tipo HTTP verso i server di C2DM.
- Deve essere capace di procurarsi e memorizzare il ClientLogin Auth Token e i Registration ID ricevuti dai dispositivi mobili.

Invio dei messaggi

In questa sezione si descrivono quali sono le operazioni che l'applicazione server deve svolgere per inviare un messaggio verso una applicazione Android. Si ricorda che, per inviare un messaggio verso una precisa applicazione, quest'ultima deve aver spedito almeno un Registration ID valido al lato server. Per inviare un messaggio, l'applicazione server effettua una richiesta HTTP di tipo POST al seguente URL: *https://android.apis.google.com/c2dm/send*, il quale rappresenta l'indirizzo che permette di raggiungere i server di Google. All'interno di tale richiesta vengono incluse le seguenti informazioni:

- **registration_id:** questo campo deve contenere il Registration ID ricevuto dall'istanza di CSAMS.

- **collapse_key:** una stringa arbitraria che viene utilizzata per identificare un gruppo di messaggi. Quando l'applicazione è offline, i server tengono in memoria tutti i messaggi appartenenti a tale gruppo; quando l'applicazione tornerá di nuovo online, solamente l'ultimo messaggio inserito nel gruppo verrá effettivamente inoltrato all'applicazione mobile. Questo accorgimento viene preso per evitare che vengano inviati molti messaggi obsoleti verso il terminale mobile. In questo caso bisogna dire peró che il servizio C2DM non garantisce ne la corretta consegna del messaggio, ne eventualmente il corretto ordinamento nel rilascio degli stessi. Per questo motivo non è detto che l'ultimo messaggio appartenente ad un gruppo per C2DM sia esattamente l'ultimo inviato dall'applicazione server.

- **il token:** all'interno del campo identificato come: *Authorization: GoogleLogin auth=[TOKEN]* deve essere inserito il valore del token che, precedentemente, l'applicazione server deve essersi procurata. Questo particolare campo viene inserito all'interno dell'HEADER della richiesta.

- **data.<key>:** contiene il payload del messaggio sotto forma di coppie del tipo nome-valore. Se presente, viene incluso all'interno dell'Intent che verrá inoltrato verso gli smartphone.

- **delay_while_idle:** se viene specificato, indica che il messaggio non deve essere inviato immediatamente nonostante il dispositivo sia attivo. I server di Google dovranno attendere per un tempo indicato in tale campo prima di inoltrare l'Intent all'applicazione destinataria.

Solamente i primi tre dei precedenti punti devono essere inseriti obbligatoriamente nel messaggio da inviare, i rimanenti due restano facoltativi. In risposta all'invio del messaggio possono arrivare diversi esiti:

- **200:** codice di risposta che sta a significare un esito positivo e che quindi la consegna del messaggio ai server di Google è avvenuta con successo. Al suo interno viene incluso un corpo contenente alcune informazioni piú dettagliate. Non è detto che la corretta ricezione del messaggio corrisponda ad una corretta formulazione dello stesso. Si potrebbero infatti presentare comunque alcuni errori, anche se il codice di risposta è stato il numero 200. In particolare gli errori possono essere dovuti al fatto che si sono inviati troppi messaggi in precedenza (C2DM impone un limite sul numero dei messaggi

da inviare in un certo lasso di tempo), oppure che all'interno del messaggio si è dimenticato di inserire qualche campo, o che è stato inserito in maniera errata, etc...

- **503:** questo codice indica all'applicazione server che i server di Google sono temporaneamente fuori servizio.
- **401:** indica che il ClientLogin AUTH_TOKEN usato nell'invio di un particolare messaggio non è valido.

Un classico schema che si deve seguire per costruire l'applicazione server è il seguente:

Listing 5.6: Applicazione server, "recupero del Token"

```
public static String getToken(String email, String password)
                        throws IOException {
                // Create the post data
                // Requires a field with the email and the password
                StringBuilder builder = new StringBuilder();
                builder.append("Email=").append(email);
                builder.append("&Passwd=").append(password);
                builder.append("&accountType=GOOGLE");
                builder.append("&source=MyLittleExample");
                builder.append("&service=ac2dm");

                // Setup the Http Post
                byte[] data = builder.toString().getBytes();
                URL url = new URL("https://www.google.com/accounts/
                    ClientLogin");
                HttpURLConnection con = (HttpURLConnection) url.
                    openConnection();
                con.setUseCaches(false);
                con.setDoOutput(true);
                con.setRequestMethod("POST");
                con.setRequestProperty("Content-Type",
                                "application/x-www-form-urlencoded");
                con.setRequestProperty("Content-Length", Integer.toString
                    (data.length));

                // Issue the HTTP POST request
                OutputStream output = con.getOutputStream();
                output.write(data);
                output.close();
```

```
28                  // Read the response
                    BufferedReader reader = new BufferedReader(new
                        InputStreamReader(
30                              con.getInputStream()));
                    String line = null;
32                  String auth_key = null;
                    while ((line = reader.readLine()) != null) {
34                      if (line.startsWith("Auth=")) {
                                auth_key = line.substring(5);
36                      }
                    }
38
                    // Finally get the authentication token
40                  // To something useful with it
                    return auth_key;
42          }
```

Listing 5.7: Applicazione server, "invio del messaggio"

```
    private final static String AUTH = "authentication";
2
            private static final String UPDATE_CLIENT_AUTH = "Update-
                Client-Auth";
4
            public static final String PARAM_REGISTRATION_ID = "
                registration_id";
6
            public static final String PARAM_DELAY_WHILE_IDLE = "
                delay_while_idle";
8
            public static final String PARAM_COLLAPSE_KEY = "
                collapse_key";
10
            private static final String UTF8 = "UTF-8";
12
            public static int sendMessage(String auth_token, String
                registrationId,
14                      String message) throws IOException {

16                  StringBuilder postDataBuilder = new StringBuilder();
                    postDataBuilder.append(PARAM_REGISTRATION_ID).
                        append("=")
18                              .append(registrationId);
```

```
            postDataBuilder.append("&").append(
                PARAM_COLLAPSE_KEY).append("=")
                    .append("0");
            postDataBuilder.append("&").append("data.payload").
                append("=")
                    .append(URLEncoder.encode(message,
                        UTF8));

            byte[] postData = postDataBuilder.toString().getBytes(
                UTF8);

            // Hit the dm URL.

            URL url = new URL("https://android.clients.google.com/
                c2dm/send");
            HttpsURLConnection
                    .setDefaultHostnameVerifier(new
                        CustomizedHostnameVerifier());
            HttpsURLConnection conn = (HttpsURLConnection) url.
                openConnection();
            conn.setDoOutput(true);
            conn.setUseCaches(false);
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-Type",
                    "application/x-www-form-urlencoded;
                        charset=UTF-8");
            conn.setRequestProperty("Content-Length",
                    Integer.toString(postData.length));
            conn.setRequestProperty("Authorization", "GoogleLogin
                auth="
                    + auth_token);

            OutputStream out = conn.getOutputStream();
            out.write(postData);
            out.close();

            int responseCode = conn.getResponseCode();
            return responseCode;
        }

        private static class CustomizedHostnameVerifier implements
            HostnameVerifier {
            public boolean verify(String hostname, SSLSession session) {
```

```
52                          return true;
                        }
54              }
```

I compiti dell'applicazione server sono divisi nalle due componenti Naming Server e Android Plugin. Infatti, il Naming server è stato modificato in modo tale da effettuare una volta per tutte la richiesta necessaria per ottenere il token e in modo tale da ricevere informazioni riguardanti i Registration ID dalle varie istanze di CSAMS. Quando una nuova richiesta di Submit viene effettuata dall'applicazione RSSIWatcher viene coinvolto per primo l'Android Plugin, chiamando il medoto "Submit" in esso contenuto. Tale metodo è stato modificato in modo da effettuare due richieste consecutive al Naming Server: una per recuperare il token e una per recuperare un determinato Registration ID tra quelli che il Naming Server stesso sta attualmente gestendo. Dopo essersi procurato questi due parametri, l'Android Plugin si preoccupa di costruire ed inviare correttamente il messaggio verso i server si Google. Lo schema seguito per effettuare le opportune modifiche ai moduli coinvolti, è stato quello esposto all'interno dei listati riportati in precedenza. In Figura 5.5 viene mostrato, in un diagramma, la sequenza di operazioni effettuate per inviare un messaggio verso CSAMS. Se si confronta questa immagine con quella in figura 3.5, si possono notare le differenze che ci sono tra la nuova e la vecchia architettura modificata in modo da supportare il servizio di C2DM.

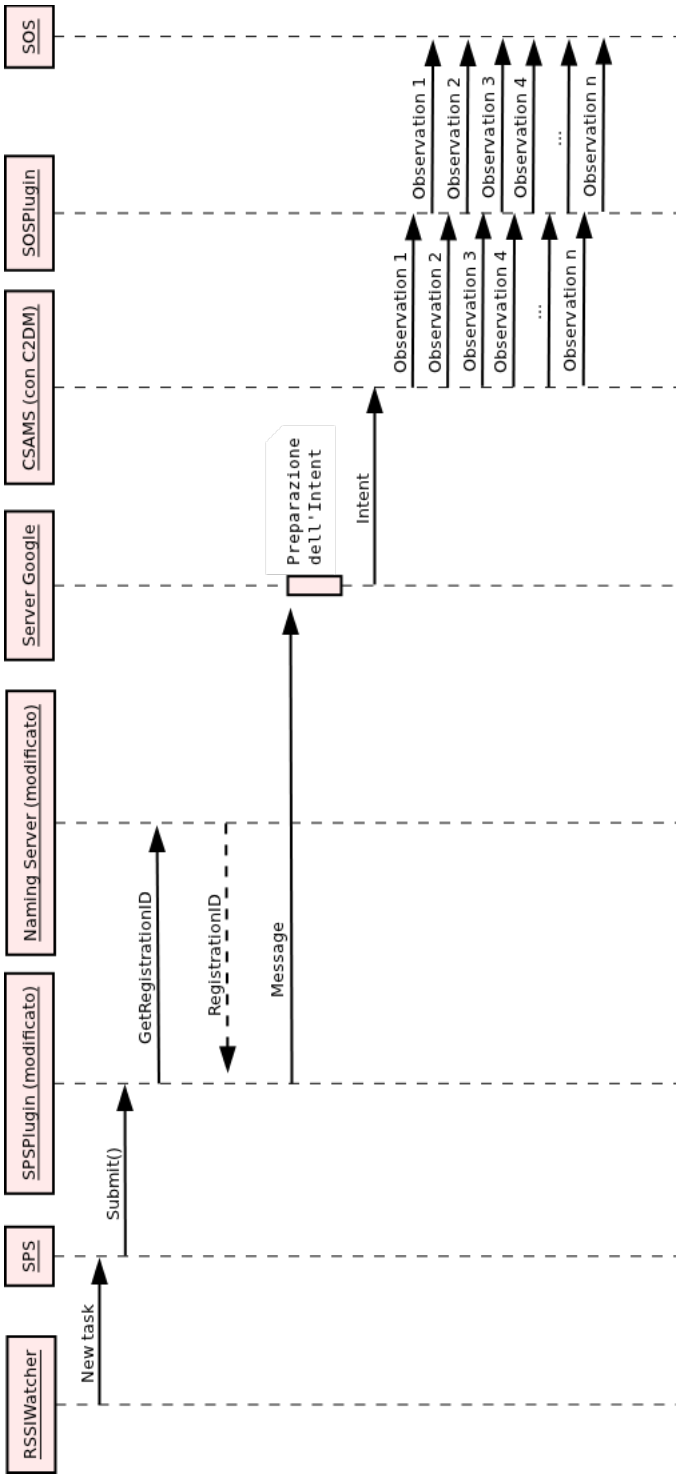

Figura 5.5: Sequenza operazioni nella nuova architettura

Capitolo 6

Conclusioni e risultati

Sfruttando un sottosistema costituito da varie parti create appositamente per gestire in maniera efficiente un insieme di sensori, è stata realizzata una applicazione web capace di sfruttare i dati forniti da tali sensori, integrati all'interno di smartphone Android, per rendere disponibile un servizio di analisi e monitoraggio della variazione del livello del segnale telefonico all'interno di una rete di telefonia mobile. L'analisi da effettuare su tali osservazione puó essere sia di tipo spaziale, andando ad osservare i dati di interesse all'interno di una mappa geografica, che di tipo temporale, andando a scegliere un area geografica di interesse per costruirvi grafici di tipo scatter. L'applicazione fornisce anche la possibilitá di interagire direttamente con i dispositivi mobili che si sono registrati per fornire un sensing partecipativo sul parametro di interesse; è possibile infatti iniziare nuovi task compilando i rispettivi campi di un semplice form con i valori della nuova campagna di osservazioni. Il sistema realizzato non prevede costi si installazione, in quanto gli unici dispositivi impiegati al suo interno sono gli smartphone posseduti dagli utenti che decidono di partecipare al sensing e un terminale sul quale è necessario installare il sistema lato server. Un altro vantaggio che si ottiene da questo sistema è la scalabilitá, la quale viene garantita grazie alla presenza dei framework offerti dalla 52North. Grazie alla grandissima diffusione dei moderni smartphone sul territorio geografico si ha inoltre un'ottima copertura geografica; tutti gli utenti possessori di smartphone possono essere infatti potenziali clienti del sistema.

Per quanto riguarda la campagna di acquisizione dati effettuata per testare l'applicazione creata, si sono lanciati per qualche giorno alcuni task. I risultati ottenuti sono mostrati in Figura 6.1, in Figura 6.2 e in Figura 6.3:

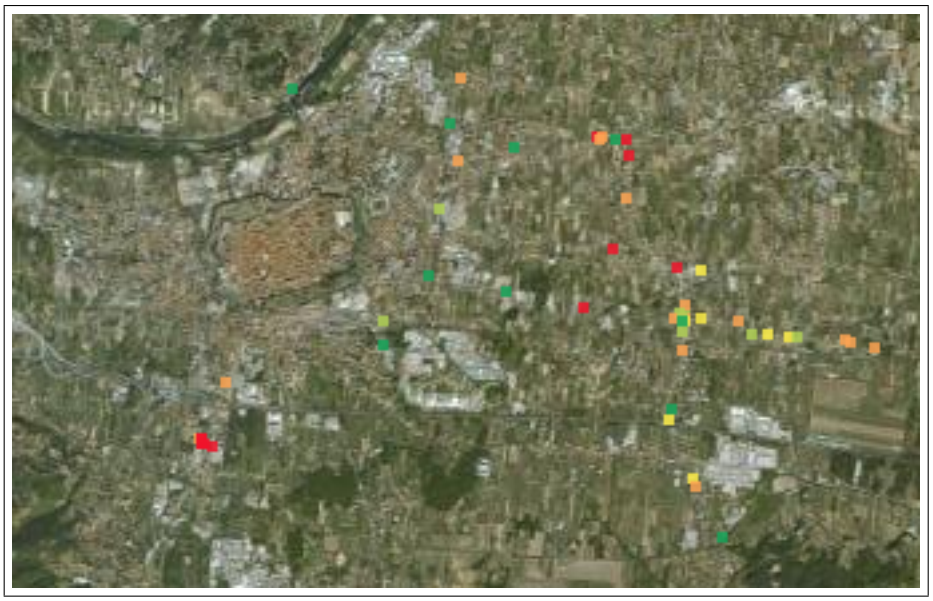

Figura 6.1: Screenshot della mappa con prima tipologia di marker

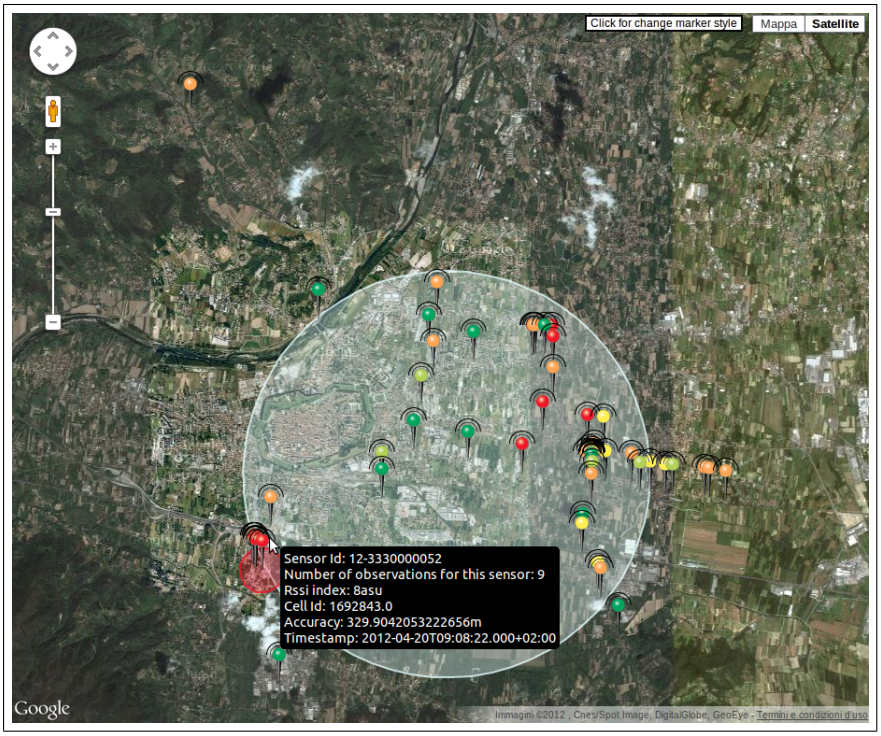

Figura 6.2: Screenshot della mappa con seconda tipologia di marker

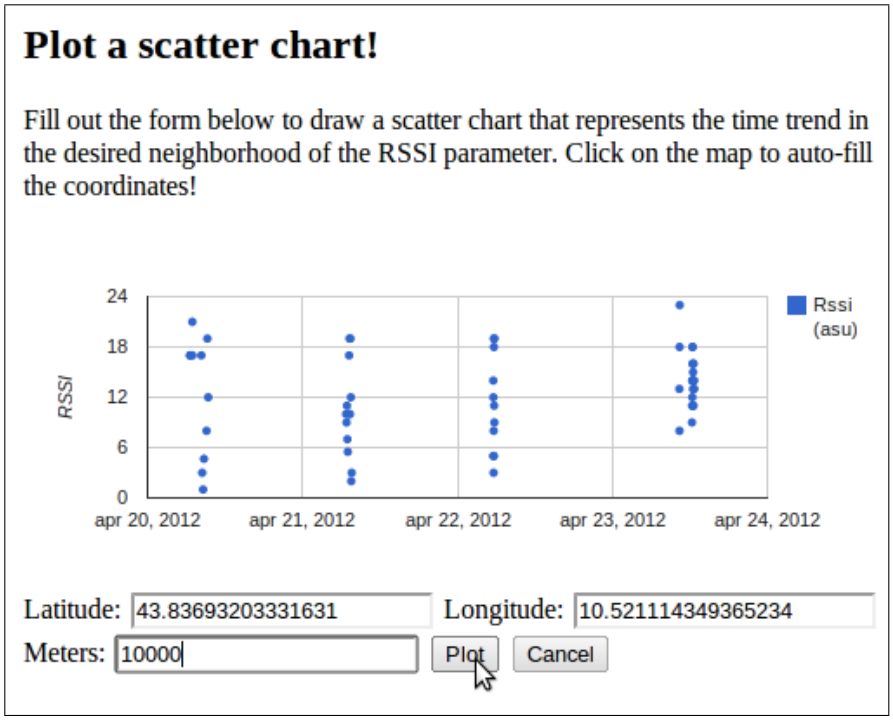

Figura 6.3: Screenshot della pagina dedicata ai grafici

Si nota come sia i grafici che la mappa con le due diverse tipologie di marker siano di facile lettura da parte di utenti che stiano utilizzando anche per la prima volta l'applicazione RSSIWatcher. Grazie al refresh della pagina, il quale puó essere effettuato in qualsiasi momento, si possono ottenere informazioni in tempo reale riguardanti sia i sensori che si sono registrati al servizio, che le osservazioni effettuate fino a quel preciso istante dai sensori integrati all'interno degli smartphones.